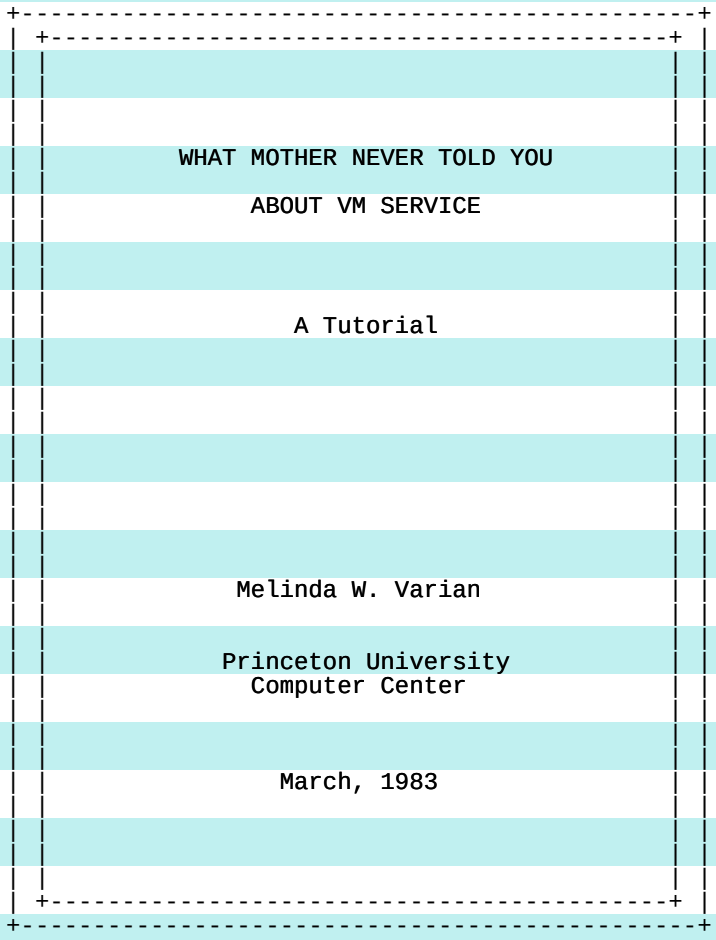

WHAT MOTHER NEVER TOLD YOU
ABOUT VM SERVICE
A Tutorial
Melinda W. Varian
Princeton University
Computer Center
March, 1983

WHAT MOTHER NEVER TOLD YOU ABOUT VM SERVICE

A Tutorial

Melinda W. Varian

Princeton University Computer Center
87 Prospect Avenue
Princeton, NJ 08544

Introduction

In talking with a number of new VM users over the past year, I have found that although they are often able to use VMSERV successfully to install the service from the VM PUTs (the preventive service tapes), they start getting into trouble when they must apply corrective service, even just taking the steps prescribed in the PUT error bucket. My impression from these conversations has been that the source of the problem is a lack of understanding of the basic VM service installation processes. These processes are not difficult to understand, but the VM education available to new users in recent years has unfortunately encouraged them to view service installation as "magic", as something which must be left in its black box. I have never been very happy having to use magic myself, especially unreliable magic. IBM's VM service magic falls into that class.

I was very fortunate when I was a new user, four years ago, to be able to sit down with a very kind and knowledgeable lady who took me through the entire service installation process, explaining it step by step. So, what I am proposing to do here today is to go through the service installation process with you, using, as she did, only the most primitive of our service primitives, in the hope of giving you a firm understanding of the entire process. Once you understand what is really going on, then it is fine to stuff it back into a black box. Indeed, you will almost certainly want to automate substantial portions of the process, using either your own or IBM's EXECs. (As I go along, I will mention the higher-level tools that are available to be used in the various steps.)

I will be presenting you with what I hope is a simple, coherent service strategy, down to the detail of disk layouts and file names. This strategy is a distillation of my own experience and the experience of a number of the VM old-timers I have talked to. It is, I hope, suitable to be picked up and used as it is by new installations. I'll go through the service process for CP first and later describe the differences in the service installation process for CMS. The important point here is that there are almost no differences. CMS service installation has a few more curlicues, but the philosophy and the mechanisms are the same. I will be using VM/SP Release 1 in my examples, but it is easy to extrapolate to the base version of VM or to BSEPP, SEPP, SP2, or HPO; the main differences are in the naming conventions. (There is an appendix in the handout which discusses the differences between installing service on SP and installing service on

other levels of VM.) I will touch briefly on the subject of installing service on other source-based VM products, but, once you know how to maintain CMS, you will have no trouble with these products.

I will not be discussing the installation of service for non-source-based VM products, such as the CMS compilers, because there is nothing I can tell you that would prepare you for doing that. Every new compiler tape is a new adventure for all of us; there are no guidelines. The best I can suggest is that VMSHARE, the network used by members of the SHARE VM Group, usually contains detailed instructions on how to fix up each new compiler tape within a couple of weeks of its appearance in the field.

The older VM installations have evolved service procedures which are remarkably similar from site to site.* There are, of course, varying views on some of the steps in the service process. I will try to point these out as I go and explain the cause of the differences. You will see, too, that there are a few areas in which the old users disagree with IBM's service installation recommendations. I will try to explain those differences, as well. Most of what IBM tells you about VM service is sound, however, so you should be sure that you are familiar with the "PSGIM", the "Field Engineering Programming System General Information" manual, G229-2228, and with the "sysgen manual", the "VM/SP Planning and System Generation Guide", SC19-6201. The best introduction to VM service installation that I am aware of is a primer written by an IBM SE, Bob Benham, which has been published by the Washington Systems Center under the title, "VM/370 Maintenance Made Simple", GG22-9277.

---

* Although some of the more outrageous views expressed in this paper are strictly my own, the paper as a whole is the result of a community effort. I set out to collect and record the folklore of the VM community on the subject of service installation. (See VMSHARE MEMO MAINT.) I was touched and pleased by the generosity with which members of the community responded. I am indebted to many VMers for taking the time to describe their service installation techniques for me, especially Bruce Marshall (ADE). I also wish to thank Kirk Alexander (PU), John Alvord (AMD), Terry Baughman (CGS), Nancy Benjamin (TD), Jim Best (PWC), Bob Cowles (CUN), Sam Drake (WSA), Jim Forkner (PSU), Lyn Hadley (IBM), John Hartmann (IBM), Pat Hennessy (HUG), Pete Jobusch (SNO), Arny Krueger (ANG), Joe Morris (UNT), Henry Nussbacher (CNY), Rich Paymer (TYM), Dick Rawson (TYM), Chuck Rodenberger (IBM), Carey Schug (CIK), Nancy Schiffmann (NII), Chris Thomas (UR), Lee Varian (PU), John Wagner (PU), Donna Walker (PY), Fred Webber (IBM), and Tom Wilson (SNO) for reviewing the manuscript and making many valuable suggestions. I am particularly grateful to Manos Maneyas (IBM) and Pat Ryall (AMD), who devoted many hours to trying to prevent me from leading new CMS system programmers astray, and to Jean Olenick (RCH), who taught me to be a VM system programmer in the first place. In all fairness, I must also thank the authors of ZORK for their unwitting contribution to this project. --MWV

Table of Contents

Appendices

I. The Use Of Control Files

Time constraints force me to assume that you have some familiarity with CMS, the CMS file system, and the common CMS tools, such as LISTFILE and EXEC. I think a brief refresher on control files and the CMS UPDATE command, as they are used in VM service installation, would be in order, though.

A. CMS UPDATE

CMS UPDATE is the tool that is used to apply fixes to VM. UPDATE is a lot like the OS utility IEBUPDTE. It can be used to insert statements into a file, delete statements from a file, or replace existing statements with new ones. UPDATE takes as input a base file, such as DMKSCH ASSEMBLE (the source for the CP scheduler), and update files, such as DMKSCH S12052DK:

FILE: DMKSCH S12052DK

```
 ./ I 4040000 $ 4045000
          NI    VMQLEVEL,255-VMCOMP  RESET COMPUTE BOUND        @VA12052
```

Note that UPDATE doesn't default to replacing the file it is updating. Its normal mode of operation is to leave the input file untouched and to build an output file containing the updated version of the input file. (This output file has the same name as the input file, but preceded by a dollar sign.) The DMKSCH S12052DK update file simply specifies that a single line of code is to be inserted into DMKSCH ASSEMBLE after the statement at sequence number 04040000 and that this line is to be given the sequence number 04045000. DMKSCH S12052DK is the IBM fix for APAR VM12052. This is the nomenclature used for VM/SP Release 1 service; the filename is the module or macro to which the fix applies, and the filetype is "S", followed by the 5-digit APAR number, followed by "DK" for CP or "DS" for CMS.

UPDATE doesn't just apply one update and it doesn't just apply all existing updates. It is much smarter than that. What it does do is use information from some other files, the "control file" and the "auxiliary control files", to decide which of the available updates are appropriate to apply to this particular system. A typical IBM-supplied control file looks like DMKSPM CNTRL, the control file for MP systems:

FILE: DMKSPM CNTRL

```
 TEXT MACS DMKSPM DMKSP DMKMAC DMSSP CMSLIB OSMACRO
 MP   UPDTMP
 AP   UPDTAP
 TEXT AUXSP12
 TEXT AUXSP11
 TEXT AUXSP
```

The first field in each of these statements is the "update level identifier", which you can think of as a name field and can ignore just now. The second field on each control file statement defines the

statement's function. There are three kinds of statements in this control file, the MACS statement, two statements which specify the name of an update ("UPDT..."), and two statements which specify the name of an auxiliary control file ("AUX..."). The MACS statement is not used by UPDATE. The other statements in this control file tell UPDATE what update files should be applied, if they exist. And the order of these statements specifies the order in which the specified updates should be applied. One tricky thing you must remember is that in applying updates you read both the control file and the auxiliary control files (the "auxfiles") from the bottom upwards.

Now, if UPDATE is told to update DMKSCH ASSEMBLE using the DMKSPM control file, it starts with the bottom-most record in the control file, "TEXT AUXSP". That "AUXSP" specifies the filetype of an auxfile. So, UPDATE looks on all accessed disks for a file named DMKSCH AUXSP, which it finds:

FILE: DMKSCH AUXSP D1

```
 S10790DK 106 UV04479 PERFORMANCE FIX FOR MAIN STORAGE OVERCOMMIT
 S12077DK 101 UV02615 RESET Q3 BIT AT Q-ADD
 S12052DK 101 UV02605 RESET COMPUTE BOUND BIT AT Q-ADD
```

(Note that if there were two such files, the one on the disk earliest in the disk search order is the one that would be used.) This auxiliary control file lists some updates that should be applied to DMKSCH, so UPDATE applies them, starting with the one at the bottom of the auxfile and working its way up. Then, UPDATE steps up a notch in the control file, to the line that says "TEXT AUXSP11". That causes it to look around for another auxfile, this one named DMKSCH AUXSP11. It doesn't find such a file, so it steps up to the next record in the DMKSPM control file, the one that says "TEXT AUXSP12". If it finds a file named DMKSCH AUXSP12, it applies any updates listed in that file. Then it steps up once more to the next record in the control file, the one that says "AP UPDTAP". This is a slightly different kind of control file record. It doesn't specify the name of an auxfile which specifies updates; instead it directly specifies the name of an update, "UPDTAP". (Think of it as being like direct addressing vs. indirect addressing.) So UPDATE looks around for an update file named DMKSCH UPDTAP and applies it if there is one. Up one more notch, to the "MP UPDTMP" record, and UPDATE looks for a file named DMKSCH UPDTMP, finds it, and applies it. With that, it is all done.

What the control file gives you, then, is a way to tailor your system. If you have an MP system, you use an MP control file, which specifies that the base source for the system is to be updated with all the APAR fixes plus all the AP and MP updates. If you have an AP system, you use the AP control file to specify that the fixes and the AP updates are to be included, but the MP updates are not. If you are running a UP, you use a control file that lists neither the AP nor the MP updates, but does list the APARs. Control files can get to be much more elaborate than this one, but even the most elaborate ones are easy, once you get the hang of it. The important thing for you to understand now is that the purpose of control files is to allow you to tailor your system.

B. VMFASM

If you have followed this much of the process, then you are almost one of the initiated. There are just two more bits of arcane lore for you to master. First, there is a tool called VMFASM which is an EXEC which is used to invoke UPDATE to apply updates and then to invoke the assembler to assemble the updated source. VMFASM uses the MACS statement in the control file to decide which maclibs to GLOBAL before doing an assembly. (You will note that the MP maclib, DMKSPM, is concatenated ahead of all the other maclibs here, since this is the MP control file.) When VMFASM invokes UPDATE, UPDATE applies the updates and then passes back to VMFASM the "update level identifier" (the "name field" in the control file record) of the highest update file that it found. Assuming in this case that there was a DMKSCH UPDTMP someplace on the accessed disks, UPDATE would tell VMFASM "MP". If there had not been a DMKSCH UPDTMP and there had been a DMKSCH UPDTAP, then UPDATE would have told VMFASM "AP". But since there was an UPDTMP, UPDATE tells VMFASM "MP", and then, after the assembly is done, VMFASM names the textfile (that's an "object deck" if you just came in from OS) DMKSCH TXTMP, rather than DMKSCH TEXT. That is, if the update level identifier of the highest-level update that was found is anything other than "TEXT", VMFASM gives the textfile a filetype formed by appending the update level identifier to the letters "TXT", as "TXTMP". Incidentally, if no updates are found, then the update level identifier from the MACS statement is used to name the textfile; that would make it "TEXT" in this case.

C. VMFLOAD

Second piece of arcane lore: there is a tool called VMFLOAD which is used to build system nuclei. VMFLOAD uses the control file, too, but it reads the control file from the top down (skipping the MACS record). The reason VMFLOAD works this way is that it is designed to pick up the version of the textfile that is tailored to your system, i.e., the one that was built with your control file. For example, if VMFLOAD is building a CP nucleus using this DMKSPM control file, when it is time to find a textfile for DMKSCH, VMFLOAD picks up the first update level identifier in this control file, "MP", and scans all the accessed disks for a file named DMKSCH TXTMP; if that fails, VMFLOAD picks up the next update level identifier and scans for DMKSCH TXTAP; and if that fails, it scans for DMKSCH TEXT. The result of using VMFLOAD with this MP control file, then, will be to build a CP nucleus which contains all the MP-specific CP code. Most CP modules don't have AP/MP updates; in those cases, VMFLOAD will use the TEXT textfile, but in the other cases, it will use the TXTAP or TXTMP textfile.

So, there you have it; that is the control mechanism for VM service installation. We use one control file per component and one or more auxiliary control files (auxfiles) for every module that has service. We do not use CDS's or FMID's or ++LMODIN statements or any of that sort of thing. Instead, it is all done with a control file and some auxfiles.

D. Recommended Control File

I am going to assume the use of the DMKSPLCL control file for CP (and parallel control files for CMS and the other products) throughout the remainder of this presentation:

FILE: DMKSPLCL CNTRL A1

```
 TEXT MACS DKLCLMAC DKPTFMAC DMKSP DMKMAC DMSSP CMSLIB OSMACRO
 LCL  AUXLCL
 PTFS AUXPTFS                      (Do not make the identifier here "PTF"!)
 TEXT AUXSP12
 TEXT AUXSP11
 TEXT AUXSP
```

DMKSPLCL CNTRL is designed to be used to install CP corrective service and local modifications on a VM/SP Release 1 system running on a uni-processor. Note that this control file would have to be modified for an AP or MP system. The MACS statement concatenates two new maclibs ahead of the IBM-supplied maclibs. One of the new maclibs, DKLCLMAC, contains local macros and locally-modified macros. The other, DKPTFMAC, contains macros which have been updated by corrective service. This control file assumes that all updates, both local and IBM-supplied, will be listed in auxfiles and that local mods and locally-applied service will be listed in auxfiles named AUXLCL and AUXPTFS. These two auxfiles are placed above the IBM-supplied auxfiles in the control file so that any corrective service will be applied on top of the preventive service from the PUTs and any local mods will be applied on top of all the IBM code. (A note of warning, don't use "PTF", rather than "PTFS" as an update level identifier; "PTF" is treated as a special case by both UPDATE and VMFLOAD.)

The philosophical basis for this control file and for the service minidisk layout that I will be recommending is:

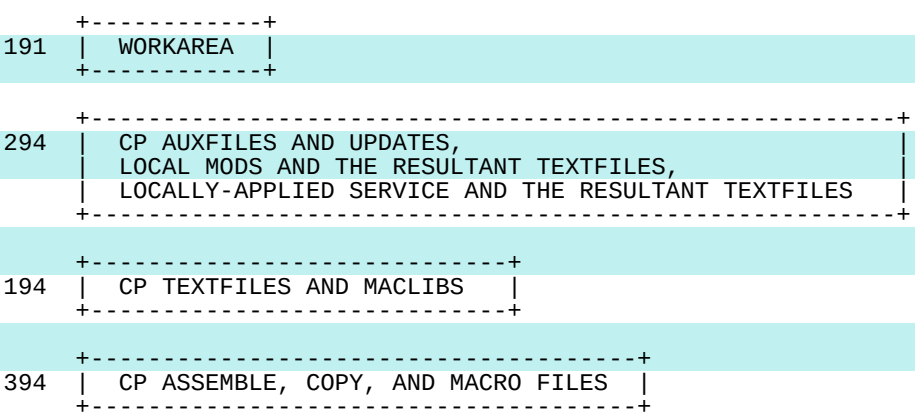
```
+--------------------------------------+
|            RULE NUMBER ONE           |
+--------------------------------------+
|  NEVER CHANGE ANYTHING IBM SENDS YOU  |
+--------------------------------------+
```

Actually, "rule" is not a strong enough word here; "dogma" would be more appropriate for the way most of the old users feel about this maxim. We do not change the files IBM sends us for the very simple reasons that we have burned ourselves when we changed their files and we have confused ourselves when we changed their files. By setting up your maintenance procedures so that you never alter any file you get from IBM, you protect yourself from your own blunders and you force some accountability on yourself. No matter how badly you mess things up, you can always get back to a known base and you can always see what IBM did and what you did. If an IBM file must be changed, then copy it to your own disk and change the copy. The changed copy will be picked up in preference to the original (because of your disk search order), but you can always get back to the original and you can always see what the differences are.

II. Service Minidisk Layouts

A. IBM Recommendation

The sysgen manual tells you to use this minidisk layout for CP service:

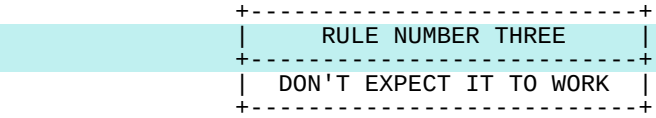


There are two things that I find really bad about this recommendation. First, mixing local mods and locally-applied service with the files from the base tape and the PUTs on the 294 disk is just bound to lead to problems. It means that you are going to have to have write access to a disk that is full of files that you should not be changing (or accidentally erasing). It is also going to make things messy and confusing going from PUT to PUT.

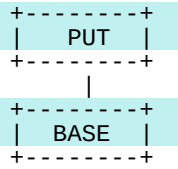
RULE NUMBER TWO
KEEP YOUR STUFF SEPARATE FROM IBM'S

Understand that in VM, corrective service is yours, not IBM's. By "corrective service", I mean locally-applied service, fixes you get from INFO/DATA or over the phone or through the mail from the Support Center or from a PUT that you haven't yet installed. I would strongly recommend that even if you decide to stick with the IBM service disk layout, you modify it to the extent of keeping all local files on your 191 disk, rather than mixing local files with the files that came on the release tape and the PUTs. (This, incidentally, is also what the SIPO/E manual suggests.)

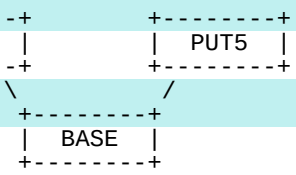
My second problem with the standard IBM service disk layout is that it violates:



When you load a service tape with VMSERV and default to the standard disk layout, all the CP textfiles, loadlists, and maclibs go onto your 194 disk, replacing textfiles, loadlists, and maclibs from the base system and from earlier PUTs. If the new PUT turns out to be a dog, you have to go through considerable contortions to back it all off. The typical old user, though, loads all the files for the CP base onto a single minidisk and never again gets a write link to that disk for the duration of that release. When he gets a PUT, he loads all of the CP files from the PUT onto a single disk and never again gets a write link to that disk for the duration of that PUT.



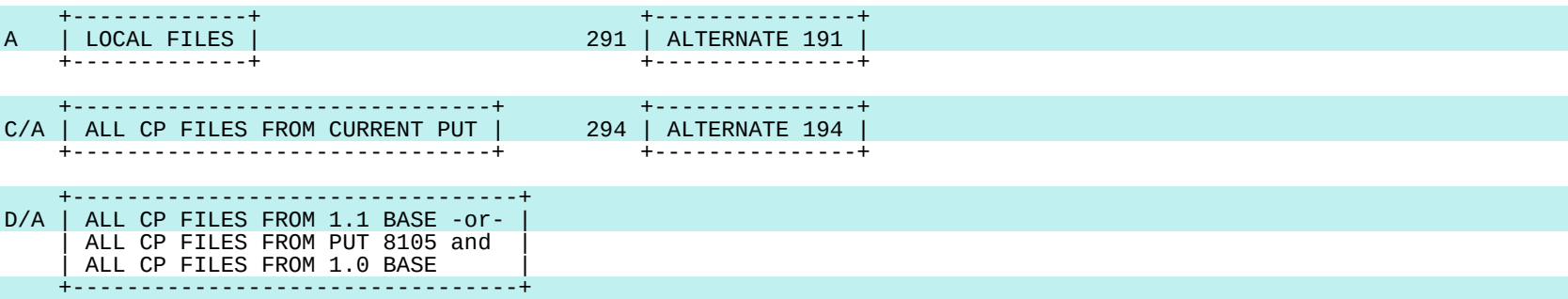
The next time he decides to install a PUT, he gets himself a new minidisk and loads all the CP files from that PUT onto that disk. (Note that VM PUTs are cumulative, so the VM user can skip over PUTs and doesn't combine the files from PUT1 and PUT5 in this example.)



With this minidisk layout, the old user has the capability of building CP systems at either of the two PUT levels, by accessing one or the other of his two PUT disks, followed by his base disk. For that matter, he can still build himself a base system, if he wants to, by accessing just the base disk. For him, backing out of a bad PUT is simply a matter of changing his PROFILE EXEC to access the old PUT disk, rather than the new one, and then rebuilding his CP nucleus.

B. My Recommendation

I recommend the use of a service disk layout similar to this one for SP CP:



This layout is typical of those used by most of the older VM installations, although many of their layouts have a few more bells and whistles. The important points about this layout are these: (1) files from the base release tape are never contaminated by service (except when there is a "level set"); (2) files from the PUT are never contaminated by locally-applied service or local mods; and (3) in going from PUT to PUT, the PUT disk and the local disk are flip-flopped with their alternates for ease of backout. Note that in this layout EVERYTHING from the PUT goes onto the PUT disk; this includes even new ASSEMBLE files. The idea is to allow you always to be able to tell for certain where a piece of questionable code came from and always to be able to back it out cleanly and reliably.

You may, of course, need to tailor this disk layout for your installation. Obviously, the virtual addresses you choose are not important, so long as you have an EXEC which can remember them and access them in the correct order. You may need a few more layers in the stack of disks. You may need to apply other vendor mods to your system, for example, in which case your other vendor service disk might be at virtual address 193 and would be accessed above the IBM PUT disk, as B/A in this example. (Incidentally, Rules Number One, Number Two, and Number Three apply to other vendors, as well as to IBM. Never change their stuff; keep it separate from your stuff and from IBM's; and don't expect it to work either.) Installations with lots of local mods frequently choose to configure a disk for base mods below their A-disk (and above the PUT disk, of course) and to keep updates to their mods on the A-disk. Other installations keep locally-applied service on a separate disk from local mods; this disk is positioned above the PUT disk and below the local mods disk.

Note that if you can't afford to keep all the source online, you can still use this sort of minidisk layout. What you should do in that case is get yourself a small minidisk for just the base source files that you actually need to use. Any time you need to apply a fix or a mod to a CP module, you should load the ASSEMBLE file for that module from the distribution tape onto your small source minidisk. (You would probably then want to leave it there for the rest of that release.) You should access this base source minidisk after all the other service minidisks.

The stack of disks can get quite deep, if you have modified mods and multiple other vendor mods disks. The depth of the stack is not a problem, though. The same principles continue to apply. Each disk in the stack contains a given class of updates and the auxfiles which list those updates and the textfiles which came with them. The stack is ordered with the files from the IBM release tape on the bottom, then the files from the IBM service tapes, then the other vendor disk (or disks), and then your stuff on top. But, any textfiles you create go on your disks. You will need an EXEC which accesses your service disks in the correct order. You may also find it convenient to leave mode B vacant so that when you have to get a TDISK to do a big assembly, you can access the TDISK as A and your 191 as B/A and not have to change any of the other accesses.

I had better explain about that 195 disk before going any further, since it represents an aberration in VM service procedures. VM PUTs have always been cumulative. Last year, however, IBM decided to do a "level set" for VM, so we got the first non-cumulative VM PUT. PUT 8105 was cumulative; it contained all the service since the SP Release 1.0 base. PUT 8106 was not cumulative, but subsequent PUTs are cumulative, in the sense that they contain all the service from 8106 on. To get current, then, we have two options. One is to build a system composed of the SP 1.0 base, plus PUT 8105 and a current PUT. The other option is to install a current PUT on top of the SP 1.1 base (which contains the 1.0 base merged with the service through PUT 8105). Note that there is no point in migrating from 1.0 to 1.1 unless your 8105 tape has gotten lost somehow.

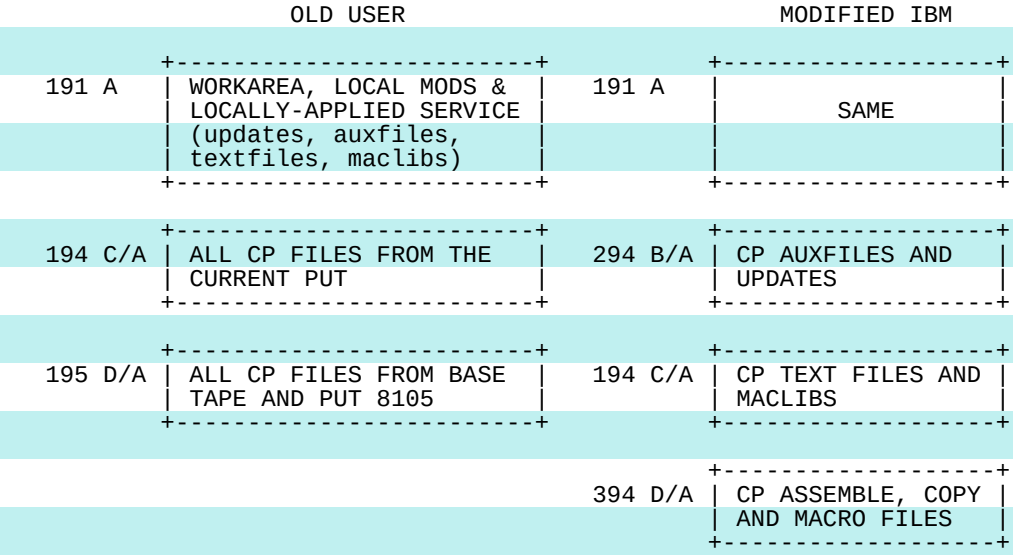
In the standard IBM service disk layout, the non-cumulative PUT did not cause much of a perturbation, because everything is always glopped together there anyhow. You install 1.0; you overlay 8105 on the same disks; and then you overlay a current PUT on the same disks. The non-cumulative PUT did perturb the old users, though, because there was no clean place for 8105 in their existing layouts. So, most old users who were already on SP when 1.1 came out opted for keeping 8105 on a separate disk accessed above the base and below the current PUT. They took this approach because overlaying 8105 on top of their base disk would have been unclean, and the alternative of loading up 8105 to the new PUT disk before loading up each new PUT would have been a pain. I keep 8105 on a separate disk myself, but we are now far enough beyond 8105 that the dust has settled, so I think that it is not unreasonable to advise you to combine 8105 with 1.0, if you aren't using 1.1.

However, IBM is about to do another level set for SP Release 1. That is why there is an entry "TEXT AUXSP12" in the recommended control file. Level sets make no sense in a system which has cumulative PUTs, but some of the people planning VM service don't understand VM service, so we have to learn to cope with such things. The level set PUT will be 8209. When you are ready to install a service level beyond 8209, you will have to glop 8209 on top of the other files on the 195 minidisk or put it on a separate disk which you access above the 195 and below the 194. Alternatively, you could start over by loading the SP 1.2 base onto your 195 disk.

One other comment about disk 195: if you use CMSBATCH, then don't use virtual address 195 for a service minidisk. CMSBATCH erases 195, so you risk absent-mindedly invoking CMSBATCH from MAINT and destroying your 195.

C. Assumptions for this Presentation

With this introduction, I am ready to start through the service process. For pedagogic purposes, I am going to assume that you are from a new VM installation. Since this is supposed to be a talk on service installation, I am going to assume that you have somehow got the SP base installed. And because I want to show you how to remove a bad fix, I will also assume that when you loaded the base, you also installed a PUT with VMSERV. I will assume at the start that you have applied no additional fixes to the system and no local mods. You have an A-disk at virtual address 191 on your MAINT userid which contains only your DMKSPLCL control file and the source and textfiles for your VM "sysgen" modules, DMKRIO, DMKSYS, and DMKSNT (and possibly DMKFCB). You have your SP CP base and service on other minidisks, either in the modified IBM layout or in my recommended old-user layout:



My specific examples will be using the old user layout but should be easy to translate to the modified IBM layout. Again, if you can't keep all the source online, then build a small base source minidisk (or use a TDISK for source) and access it after all the other service disks, as 196 E/A, in this case. All my examples will expect the service disks to be accessed either as read-only extensions of the A-disk, as shown here, or as read-only extension of themselves, unless some other access is explicitly stated. (You may prefer not to make the service disks extensions of your A-disk, because COPYFILE behaves better if you don't.) You want never to have write access to your PUT and base disks except when you are actually loading tapes onto them. There are two reasons for this; it will keep you from altering files that shouldn't be altered, and it will keep the assembler from playing strange tricks on you and writing files where you don't expect them to be written.

III. Installing Corrective Service for CP

A. Making a CP Module Resident

You get the system into production and you go along for a while and then CP crashes with a PRG006 abend. You call the Support Center and tell them what happened and they tell you it's a known problem. The APAR number is VM12989. There is no fix yet, but they tell you that the circumvention is to make the CP module DMKVMC resident. They assume that you know how to do that.

Here is how it is done. CP modules are either permanently resident or pageable. This is determined by their position in the "loadlist". The loadlist is an EXEC file which simply lists all CP nucleus modules in the order in which they are to be placed in the nucleus. IBM supplies several different load lists; you choose the one you need for your configuration:

```
APLOAD   EXEC     AP/MP systems
AVLOAD   EXEC     AP/MP systems with a V=R area
CPLOAD   EXEC     UP systems
CPLOADSM EXEC     Small UP systems
VRLOAD   EXEC     UP systems with a V=R area
```

All the CP loadlists look something like this:

```
+----------------------------+
|  &CONTROL OFF              |
|  &1 &2 &3 DMKLD00E LOADER  |
|  &1 &2 &3 DMKPSA           |
|     .                      |
|     .                      |
|  (other resident modules)  |
|     .                      |
|     .                      |
|  &1 &2 &3 DMKCPE           |
|  *                         |
|  *    END OF THE RESIDENT  |
|  *    VM/370 NUCLEUS       |
|  *                         |
|  &1 &2 &3 DMKTAP           |
|  &1 &2 &3 DMKVMD           |
|  &1 &2 &3 DMKCFC           |
|     .                      |
|     .                      |
|  &1 &2 &3 DMKVMC           |
|     .                      |
|     .                      |
|  &1 &2 &3 DMKURS           |
|  &1 &2 &3 DMKCKP           |
|  &1 &2 &3 LDT    DMKSAVNC  |
+----------------------------+
```

As you can see, a loadlist is just a list of filenames and, in some cases, filetypes. The VM loader (DMKLD00E LOADER) is the first item in the loadlist; both its filename and its filetype are specified. The loader is followed by DMKPSA (page zero of the CP nucleus) and the other resident modules, ending with DMKCPE ("CP end"). After DMKCPE come the pageable modules. The filetypes are not specified for the system modules because the loader uses your control file to decide what the filetypes should be (TEXT, TXTPTFS, TXTLCL, etc.). The last two items in the loadlist must be DMKCKP (the checkpoint module) and a file named LDT DMKSAVNC, which contains one record, which says "LDT DMKSAVNC". This is a LOADER TERMINATE statement which defines the entry point to be DMKSAVNC, the code which actually writes the new nucleus out to the sysres volume.

To make a pageable module resident, all you do is move it from being after DMKCPE in the loadlist to being before DMKCPE. You could do this by just editing the loadlist EXEC that you are using. That is fast and easy, but it is also slovenly, and it will get you into trouble someday. The loadlists should be maintained with UPDATE and auxfiles. Here is how it is done. First, create an update file to sequence the loadlist EXEC you are planning to use. (This is made necessary by the fact that IBM maintains the loadlists with the editor, rather than with UPDATE, so it doesn't sequence them. You will be glad to know that not using UPDATE to maintain the loadlists gets THEM into trouble, too.) To sequence CPLOAD EXEC, for example, you would build an update file called CPLOAD SEQUENCE:

FILE: CPLOAD SEQUENCE A1

```
./  S
```

Now, say you are using the CPLOAD loadlist and you want to make DMKVMC resident as the circumvention for APAR VM12989. Create an auxfile named CPLOAD AUXPTFS which lists that APAR and your SEQUENCE update:

FILE: CPLOAD AUXPTFS A1

```
 *$*$*$*$*$*$*$*$*$*$*$*$*$*$*$ AUXPTFS $*$*$*$*$*$*$*$*$*$*$*$*$*$*$*$*$*
 S12989DK - mm/dd/yy - MAKE DMKVMC RESIDENT TO CIRCUMVENT PROBLEM IN APAR
 *          VM12989 - PRG006 ABEND - DUMP PRB00001 - PER SUPPORT CENTER
 SEQUENCE - mm/dd/yy - SEQUENCE LOADLIST EXEC BECAUSE IBM WON'T
```

Then invoke XEDIT with the CTL option:

```
 xedit cpload exec c ( ctl dmksplcl
```

XEDIT will first use your CPLOAD AUXPTFS file to update CPLOAD EXEC. It will pick up the bottom-most update, CPLOAD SEQUENCE, and apply that, thus sequencing CPLOAD EXEC. Then it will look for the next update listed in the auxfile, CPLOAD S12989DK. Since that doesn't yet exist and since you specified the CTL option, XEDIT will assume that what you really want to do in this XEDIT session is to create the update CPLOAD S12989DK. It will put you into an XEDIT session with the updated (sequenced) CPLOAD EXEC. In XEDIT, you should move the loadlist entry for DMKVMC to just before the entry for DMKCPE and then enter the XEDIT command "file". This will create an update file named CPLOAD S12989DK:

FILE: CPLOAD S12989DK A1

```
 ./ I 00089000           $ 89100 100
 &1 &2 &3 DMKVMC
 ./ D 00181000           $
```

Assuming that IBM's CPLOAD EXEC is on your C-disk, which is accessed read-only, you build a modified CPLOAD EXEC on your A-disk with this command:

```
 update cpload exec c dmksplcl ( ctl rep
```

The CTL option simply tells UPDATE that DMKSPLCL is a control file, rather than an update file. Literally, the REP option tells UPDATE to replace the file being updated; in this case, however, since UPDATE will be putting the updated file on your A-disk, it will not replace the original on your read-only C-disk, but, because you said REP, it will name the updated file CPLOAD EXEC, rather than $CPLOAD EXEC, so you don't have to rename it.

When you build your CP nucleus, VMFLOAD will use this modified version of CPLOAD EXEC from your A-disk, rather than the original one on your C-disk, because it will find the one on the A-disk first. So, you will get a nucleus which has DMKVMC resident, rather than pageable, and you should get fewer PRG006 abends.

There are a number of different approaches to building and delivering a CP nucleus. IBM provides you with two different EXECs for doing it, VMSERV and GENERATE. Of the two, GENERATE has more function and flexibility. I will discuss the various theories of CP nucleus delivery later, but right now, let's do it a very simple (and not very safe) way, by hand. You are logged onto MAINT with a virtual machine size of at least 512K; you have your service minidisks accessed in one of the standard ways; and MAINT's directory entry includes a write link to your system residence volume, with the virtual address the same as the real address. You issue these commands:

```
 close pun
 close rdr
 spool pun to * class n
 spool rdr class n

 vmfload cpload dmksplcl        (CPLOAD might not be the loadlist you need)
 SYSTEM LOAD DECK COMPLETE
 PUN FILE 4444  TO  MAINT    COPY 001   NOHOLD

 order rdr 4444

 ipl 00c clear
 NUCLEUS LOADED ON VMR901 --- STARTING CYL/BLK=004, LAST CYL/BLK USED=005
 CP ENTERED; DISABLED WAIT PSW '00020000 00000012'

 spool prt to ipcs         (or whatever you call your dump-reading machine)
 close prt
 PRT FILE 4448  TO  IPCS     COPY 001   NOHOLD
```

You spool your virtual punch to your virtual reader and you tidy things up by closing your reader and your punch and spooling them both class N (for "nucleus"). You invoke VMFLOAD, which steps through your loadlist EXEC, punching a textfile ("object deck") for each of the items in the loadlist. Because you have your virtual punch spooled to yourself, these virtual decks will end up in your virtual reader. When VMFLOAD is done punching all the files in the loadlist, it puts out the message, "SYSTEM LOAD DECK COMPLETE". You make sure that this system load deck is the first file in your virtual reader and then you IPL your virtual reader, 00C. This brings in the loader, which was the first item in the loadlist. The loader reads in the textfiles which were behind it in the reader file, "link-edits" them into a CP nucleus, and passes control to DMKSAVNC, which writes the nucleus onto the volume described by the SYSRES statement in the DMKSYS textfile.

The use of a "standalone" loader and all this punching and reading of virtual cards may strike you as a rather quaint way of building a system, but it works well and quickly, and you now have a new CP nucleus on your sysres volume. While the loader was building your nucleus, it was also creating a load map, which it sent to your virtual printer. Since your virtual printer was spooled to your IPCS virtual machine, IPCS will be able to read the loadmap from its virtual reader onto a minidisk. This map should be archived for use in reading CP dumps.

B. Removing a Bad Fix

At some point, this new nucleus gets IPLed and you run fine for a couple of days, but then you notice that performance seems to be degrading. You ask SMART (the VM Real Time Monitor) if anything is wrong, and SMART points out that your system is "extended" 120 pages. This means that the real memory being used for CP control blocks has grown 120 pages beyond the allocation you specified in the SYSCOR parameter in DMKSYS. In other words, 120 page frames that should be being used for user pages are not available to the users. Not surprisingly, the INDICATE LOAD command shows that your page steal rate is way up. Looking at SMART's log, you see that the number of extended pages has been increasing steadily over the past couple of days since your last IPL. So, you call the Support Center. Again, they tell you that you have encountered a known problem. The APAR is VM14782. There is no fix yet, so you ask to be put on the "interested parties" list for VM14782, so that you will be notified as soon as the fix is available. (They will not send you the fix, but they will send you a letter saying that the fix exists. You might think that it would be more useful if they sent you the fix rather than the letter, but those letters are potentially valuable. A proposal has been made that the SHARE VM Group should award a prize each year to the person receiving the most APAR-closing letters.) But, in the meantime, you need to do something about your performance degradation. You ask them about that and they tell you that the problem is caused by a "PE", a "PTF in Error", a bad fix. They suggest that you might want to remove this PE, APAR VM12684, from your system. They assume that you know how to do that.

Here is how you remove that PE. The first thing to do is to find out what modules and macros were hit by VM12684:

```
 listfile * s12684dk *

 DMKMSG   S12684DK C1
 R;
```

It turns out that no macros were updated by VM12684 and only one module was, DMKMSG. This is good, because it means you don't have to change any maclibs. But you do want to change DMKMSG.

One point that needs to be made clear for the benefit of those of you who have recently come into VM from VS is that VM fixes are never installed permanently until the developers incorporate them in the source for a new release of the system. That is, in the jargon, VM has "fixed-base", rather than "moving-base", maintenance. Therefore, when we speak of removing a bad fix, what we are really talking about is not applying the bad fix. In VM, you remove a bad fix by using VMFASM to update a temporary copy of the base source file (the ASSEMBLE file) with all the updates except the bad one and then assemble the updated copy. The updated source file is erased as soon as the new textfile has been created, thus leaving the original source file unaltered.

This PE is in your system because at some point IBM listed it in an auxfile for DMKMSG. So, to get this PE out of your system, all you have to do is take its name out of that auxfile, reassemble DMKMSG, and rebuild your CP nucleus, incorporating your new textfile for DMKMSG. If you are running an SP Release 1.0 system, you will have an IBM auxfile called DMKMSG AUXSP. If you have also got some PUT beyond 8105 installed, you will also have an auxfile called DMKMSG AUXSP11. If you are running SP Release 1.1, you will have only DMKMSG AUXSP11, because the APARs listed in the AUXSP auxfiles were merged into the Release 1.1 base. (They did this to make things simple for us.) Anyhow, use LISTFILE to find whatever DMKMSG auxfiles you have and look to see if you can find one which specifies the update for VM12684, which is called S12684DK, of course. It turns out that it is in the AUXSP11 auxfile for DMKMSG:

FILE: DMKMSG AUXSP11 C1

```
 S13491DK 108 UV05121 CORRECT HANDLING OF START FIELDS IN MESSAGES
 S12594DK 108 UV04922 IMBEDDED LINEND CHARACTER HANDLED INCONSISTENTLY
 S12684DK 107 UV04645 DMKDGD RCWTASK DECREMENTED AS A TIMER VALUE.
```

Since what you want to do is take this entry out of the auxfile, you could just edit the auxfile, but that would violate Rule Number One. Instead, copy DMKMSG AUXSP11 from your C-disk to your A-disk and edit the copy on your A-disk. Don't delete the S12684DK line; "comment" it out instead; that is, put some asterisks in position 1, so that UPDATE will treat the entry as a comment, rather than as an update name. And don't just comment it out; add a comment under the entry telling when you did it, why you did it, who told you to do it, what the APAR number is, what your incident number is, what your dump number is, what the name is of any VMSHARE file in which the problem is discussed, and anything else you can think of that might jog your memory later on.

```
                        +----------------------+
                        |   RULE NUMBER FOUR   |
                        +----------------------+
                        |  ALWAYS LEAVE TRACKS |
                        +----------------------+
```

A nice little touch is to add a "title" line at the top of the modified auxfile, which will come out in your assembly listing and make it clearer what is going on.

FILE: DMKMSG AUXSP11 A1

```
 *$*$*$*$*$*$*$*$*$*$*$*$* MODIFIED AUXSP11 $*$*$*$*$*$*$*$*$*$*$*$*$*$*$*
 S13491DK 108 UV05121 CORRECT HANDLING OF START FIELDS IN MESSAGES
 S12594DK 108 UV04922 IMBEDDED LINEND CHARACTER HANDLED INCONSISTENTLY
 *** S12684DK 107 UV04645 DMKDGD RCWTASK DECREMENTED AS A TIMER VALUE.
 *** S12684DK REMOVED BECAUSE OF CORE CANCER - 12/01/81 - VM14782 - MWV
 ***          PRB00002 - SEE VMSHARE 'PROB 1SP07'
```

You will note that I didn't hesitate to tell you to remove the PE, even though the auxfile lists two other fixes which are to be applied on top of this fix. It may be that the fix you're removing is a "pre-requisite" for one of those two fixes, in which case, you've got a problem. However, experience has shown me that when I remove a fix this way, if I don't get sequence error messages from UPDATE or syntax error messages from the assembler, then it is very unlikely that the fix I'm removing is a functional pre-requisite for the other fixes listed above it in the auxfile. This is one of the joys of maintaining a system that has source-level service. Some IBMers will advise you instead to remove all the fixes above the PE in the auxfile, but then you run the risk of removing a fix you really need. There is, of course, an exposure which ever way you choose. My advice, though, is to leave the other updates in the auxfile as long as neither UPDATE nor ASSEMBLE complains, unless you actually have some reason to believe that the PE really is a pre-requisite for them. In either case, whenever you remove an APAR update from a module or macro, be certain that you remove the corresponding updates from all the modules and macros hit by the APAR. Leaving half a fix in your system is a sure way to get into trouble.

You now have two files named DMKMSG AUXSP11, the original one on your C-disk and the modified one on your A-disk. The one you want to use is, of course, the one on your A-disk. This will happen automatically, because of your disk search order. To reassemble DMKMSG without incorporating the PE, you use VMFASM:

```
vmfasm dmkmsg dmksplcl

UPDATING 'DMKMSG ASSEMBLE D1'
APPLYING 'DMKMSG S11690DK D1'
APPLYING 'DMKMSG S11792DK D1'
APPLYING 'DMKMSG S09531DK D1'
APPLYING 'DMKMSG S13028DK D1'
APPLYING 'DMKMSG S12594DK C1'
APPLYING 'DMKMSG S13491DK C1'
FILE 'DKLCLMAC MACLIB' NOT FOUND.
FILE 'DKPTFMAC MACLIB' NOT FOUND.
ASMBLING DMKMSG
ASSEMBLER (XF) DONE
NO STATEMENTS FLAGGED IN THIS ASSEMBLY
DMKMSG TEXT CREATED
R;
```

This VMFASM command causes a copy of DMKMSG ASSEMBLE D1 (the base source file) to be updated with the four fixes listed in DMKMSG AUXSP D1 and the two remaining fixes listed in DMKMSG AUXSP11 A1. Note that the update DMKMSG S12684DK doesn't get applied. Fixes listed in DMKMSG AUXPTFS * and DMKMSG AUXLCL * would also be applied, if those auxfiles existed. The updated file is assembled, and an object deck named DMKMSG TEXT is written onto your A-disk. (The updated ASSEMBLE file is erased before VMFASM exits.) Because of your disk search order, when you build your new CP nucleus, the DMKMSG TEXT from your A-disk will get used, rather than the bad DMKMSG TEXT that IBM sent you, which is on your C-disk.

If you type or edit that DMKMSG textfile, by the way, you will see one of the really sexy features of VM service. The beginning of this "object deck" lists precisely what fixes have been applied and even includes those comments that you put into the auxfile. The point here is that every textfile in a VM system is completely self-identifying. To find out what fixes a given textfile contains, you need only type it:

FILE: DMKMSG TEXT A1

```
 S11690DK 101 UV02593 MSDVHDIR206E SMSG RC=01 AFTER APPLYING VM09994
 *         DMKMSG   S11690DK D1 MNT195  10/21/80   14:05:00
 S11792DK 101 UV02626 CANNOT FORCE USER OFF WHEN IN ECHO COMMAND
 *         DMKMSG   S11792DK D1 MNT195  10/21/80   14:05:00
 S09531DK 101 UV02701 ABENDLOK001 AFTER SWTCHVM MACRO FOR FRETTED VMBLOK
 *         DMKMSG   S09531DK D1 MNT195  10/24/80   15:33:00
 S13028DK 105 UV04184 VM10995 CREATES FUTURE EXPOSURE
 *         DMKMSG   S13028DK D1 MNT195  03/13/81   09:25:00
 *$*$*$*$*$*$*$*$*$*$*$*$*$* MODIFIED AUXSP11 $*$*$*$*$*$*$*$*$*$*$*$*$*$*$*
 *** S12684DK 107 UV04645 DMKDGD RCWTASK DECREMENTED AS A TIMER VALUE.
 *** S12684DK REMOVED BECAUSE OF CORE CANCER - 12/01/81 - VM14782 - MWV
 ***           PRB00002 - SEE VMSHARE 'PROB 1SP07'
 S12594DK 108 UV04922 IMBEDDED LINEND CHARACTER HANDLED INCONSISTENTLY
 *         DMKMSG   S12594DK C1 MNT194  08/25/81   07:18:01
 S13491DK 108 UV05121 CORRECT HANDLING OF START FIELDS IN MESSAGES
 *         DMKMSG   S13491DK C1 MNT194  08/25/81   07:18:01
 *         DMKSP    MACLIB   A1 MNT194   8/31/81   10:45:01
 *         DMKMAC   MACLIB   A1 MNT194   7/14/81   10:46:59
 *         DMSSP    MACLIB   S2 CMS190   6/10/81   18:45:00
 *         CMSLIB   MACLIB   S2 CMS190   2/07/81   12:19:00
 *         OSMACRO  MACLIB   S2 CMS190   3/18/81   13:32:56
 *         DMKMSG   ASSEMBLE A1 MNT195   7/31/80   19:45:00
  ESD            DMKMSG          DMKCVTDB        DMKCVTBD
   .
   .
   .
  END                            15741SC103 020182003
```

When the loader loads such a textfile, it strips off all this non-executable commentary and puts it into the loadmap.

I should point out that there is another workable mechanism for removing a bad fix, other than commenting it out in a copy of IBM's auxfile. You can accomplish the same thing by building a dummy update on your A-disk and giving it the same name as the bad update you wish to remove. When you invoke VMFASM, your dummy update will get picked up in preference to IBM's bad update, which is on a disk later in the search order. For example, you could have removed VM12684 by creating the dummy update DMKMSG S12684DK on your A-disk and reassembling DMKMSG:

FILE: DMKMSG S12684DK A1

```
 ./ * DUMMY UPDATE TO REMOVE VM12684 BECAUSE OF CORE CANCER
 ./ * mm/dd/yy - VM14782 - PRB00002 - SEE VMSHARE 'PROB 1SP07'
```

The only argument against doing it this way is that you don't end up with your comments about removing the fix in the textfile and the loadmap. If you look carefully at the textfile and the loadmap, however, you will see that the update came from your A-disk, rather than from one of IBM's disks. It is a matter of taste. The important point is to leave tracks.

C. Applying a Fix

You build and install another new CP nucleus, which now has DMKVMC resident AND has VM12684 removed. Things go along smoothly for a few hours and then the director of your computer center comes looking for you to tell you that his terminal is hung up. He can't do anything with it. Thinking all you need to do is show him the CLEAR key, you go take a look, but you find that you can't do anything with it either. So, you log onto a privileged userid and issue a FORCE command. The system says it forced him, but he is still logged on, and his terminal is still hung up. You have just met your first "hunguser", the first of many. This particular hunguser is getting a trifle upset about the report he is trying to generate, so you tell the operators to take the system down with a dump.

You try looking at the dump, but DUMPSCAN will not display most of the things you want to look at; it keeps telling you that there is no page zero in the dump, and, of course, all the pointers it needs are in page zero. After pounding your head against the wall for a while, you discover that the reason there is no page zero in the dump is that someone had put it on the free list, and pages on the free list don't get dumped. You call the Support Center and open two incidents, one for page zero not getting dumped and one for the hunguser problem. Neither of these seems to be a known problem, so you are asked to mail in the dump and your nucleus loadmap on tape, which you do. You wait a while and get three more hungusers, so you call back and raise the severity of the problem. Then, despite your having sent them an almost useless dump, one of the guys at Second Level figures out that your hunguser problem is a known problem and is fixed by APAR VM13318. He calls you and reads you updates to the IOBLOKS copy section and DMKIOS. He assumes you know what to do from there.

Here is what you do. First, you have to key in the two update files he read you. That is, you have to key in the two update files unless you already have them in machine-readable on a PUT which you have received but have not yet installed. To find out whether the fix is on the latest PUT, you can either look at the SP "Memo to Users" from that PUT, if you have already printed it off, or you can simply mount the tape and try to load the fix you need:

```
 vmfplc2 load * s13318dk ( eot
```

Let's assume that you don't luck out; it's not on the PUT and it's not in INFO/DATA either, so you have to key it in. The two files you create should be named IOBLOKS S13318DK and DMKIOS S13318DK. The updates should be marked with an "update identifier" field somewhere on the right hand side. I find that it is useful to make this field NOT look just like it will when IBM sends the fix out on a PUT, so I always know at a glance that this is a fix that I keyed in and, thus, that it may very well be different from the final version of the fix. I just put the five-digit APAR number in column 67:

```
FILE: IOBLOKS S13318DK A1

 ./ R 00400000 00400000 $ 00400000 1000                              13318
 *              |       IOBQDTO         | I*9 |  IOBRSV4        |  13318
 ./ R 00790000 00790000 $ 00790000 1000                              13318
 IOBERCNT DS    1X        I*9  ERROR RETRY COUNT                     13318
 IOBRSV4  DS    3X             RESERVED FOR FUTURE USE               13318

FILE: DMKIOS S13318DK A1

 ./ R 15700000 15710000 $ 15700000 200                               13318
          BZ    IOSCCH          NO LOGOUT PEND - CALL DMKCCH         13318
          SLR   R1,R1           CLEAR WORK REG                       13318
          IC    R1,IOBERCNT     LOGOUT PEND / CCC RETRY COUNT        13318
          C     R1,F10          RETRY COUNT EXCEED 10                13318
          BH    IOSFATAL        YES, REFLECT TO USER                 13318
          LA    R1,1(,R1)       INCREMENT RETRY COUNT                13318
          STC   R1,IOBERCNT     STORE IN IOBLOK                      13318
          B     IOSQBSY         QUEUE OFF THE CHANNEL                13318
 IOSFATAL NI    RCHSTAT,255-RCHBUSY TURN OFF CHANNEL BUSY            13318
          NI    RCUSTAT,255-RCUBUSY TURN OFF CONTROL UNIT BUSY       13318
          NI    RDEVSTA4,255-RDEVBUZY DEVICE NO LONGER BUSY          13318
          XC    RDEVAIOB,RDEVAIOB CLEAR ACTIVE IOBLOK                13318
          XC    RDEVPROC,RDEVPROC CLEAR PROC ADDRESS                 13318
          OI    IOBSTAT,IOBFATAL FATAL I/O IF EXCEED 10 RETRIES      13318
          CALL  DMKSTKIO          STACK THE IOBLOK                   13318
          B     IOSTEXIT         EXIT TO CALLER                      13318
 IOSCCH   NI    RDEVSTA4,255-RDEVBZCH RESET DEVICE BUSY ON CHAN      13318
```

Next you need to create AUXPTFS auxfiles for IOBLOKS and DMKIOS. Here is what they should look like:

```
FILE: IOBLOKS AUXPTFS A1

 *$*$*$*$*$*$*$*$*$*$*$*$*$*$*$ AUXPTFS $*$*$*$*$*$*$*$*$*$*$*$*$*$*$*$*
 S13318DK - 10/06/81 - FROM G.B., LEVEL II, FOR PRB00003 - HUNGUSERS

FILE: DMKIOS AUXPTFS A1

 *$*$*$*$*$*$*$*$*$*$*$*$*$*$*$*$*$*$*$ AUXPTFS $*$*$*$*$*$*$*$*$*$*$*$*$*$*$*$*$*
 S13318DK - 10/06/81 - FROM G.B., LEVEL II, FOR PRB00003 - HUNGUSERS
```

This is the first time you have had to update a system macro, so we had better talk some about maclibs. IBM provides two macro libraries for SP CP, DMKMAC MACLIB and DMKSP MACLIB. DMKMAC contains the macro and copy sections which are not "versioned" by SP. DMKSP contains the macro and copy sections which are "versioned" by SP. Nobody knows why IBM didn't merge the macro libraries when they merged the source, but they didn't, so you need to be conscious of the fact that there are two libraries. (Similarly, there are two CMS maclibs, CMSLIB and DMSSP.) There are two schools of thought concerning the best way to apply corrective service to system macros. One acceptable approach is to copy the IBM maclib to your A-disk and replace the macro in that copy. Unfortunately, IBM doesn't provide a nice primitive for doing that, though it would be easy enough for

you to write your own EXEC for this, using VMFASM as a guide. What your EXEC should do is update the macro using UPDATE and the specified control file, replace the macro in the specified maclib using the MACLIB REP command, append the UPDATES file to the maclib using COPYFILE, and erase the updated macro, the UPDLOG file, and the UPDATES file. If you are not going to be applying much corrective service and you are not going to be modifying the system much, then this would be the soundest approach for you to use in applying corrective service to macros.

The second approach is to build a separate maclib for macros and copy sections which have corrective service applied to them. A reasonable name might be PTFMAC MACLIB, but you may also want a service maclib for CMS, so I would suggest DKPTFMAC and DSPTFMAC and parallel libraries for macros and copy sections which are updated by local modifications called, say, DKLCLMAC and DSLCLMAC. Don't name your own maclib NEWMAC; VMFMAC uses NEWMAC MACLIB as a temporary file. Note that if you are going to build your own maclibs, you must include their names in the MACS statement in your control file, as I did in the recommended control file. Note, too, that you can't list more than eight maclibs in a control file, because that is all that the GLOBAL MACLIB command can handle.

You build a maclib by creating an EXEC file whose filename is the same as the filename of the maclib and whose contents are a list of the macros and copy sections that you want included in the maclib. You can take a look at DMKMAC EXEC to see how this is done. In this case, since we have only one item to go into DKPTFMAC MACLIB, the maclib EXEC looks like this:

FILE: DKPTFMAC EXEC

```
 &1 &2 IOBLOKS                                                  13318
```

Once you have this EXEC built, you can build the maclib with the VMFMAC command:

```
 vmfmac dkptfmac dmksplcl

 UPDATING 'IOBLOKS COPY D1'
 APPLYING 'IOBLOKS S11854DK D1'
 APPLYING 'IOBLOKS S12470DK D1'
 APPLYING 'IOBLOKS S12635DK C1'
 APPLYING 'IOBLOKS S13318DK A1'
 IOBLOKS COPY ADDED.
 DKPTFMAC COPY ADDED.
```

VMFMAC is very similar to VMFASM and VMFLOAD. It steps through the maclib EXEC, putting each of the listed macros or copy sections into the new maclib, after updating it as specified by the control file.

One drawback to creating these additional maclibs, rather than modifying the ones you get from IBM, is that some of the installation EXECs for program products know the names of the standard maclibs and will have to be changed to know about your new maclibs as well. I find, however, that the program product installation EXECs cannot be trusted to have the standard maclib names right anyhow, so either way you are going to have to check them and possibly fix them up.

Once you have your PTF maclib built, you are ready to reassemble DMKIOS. You will be using our DMKSPLCL control file, so VMFASM will GLOBAL the maclibs listed in the MACS record in that control file. This will concatenate DMKSP MACLIB and DMKMAC MACLIB after your DKPTFMAC MACLIB, so the assembler will use the updated version of IOBLOKS.

```
vmfasm dmkios dmksplcl

UPDATING 'DMKIOS ASSEMBLE D1'
APPLYING 'DMKIOS S11399DK D1'
APPLYING 'DMKIOS S11412DK D1'
APPLYING 'DMKIOS S11946DK D1'
APPLYING 'DMKIOS S11854DK D1'
APPLYING 'DMKIOS S12022DK D1'
APPLYING 'DMKIOS S12293DK D1'
APPLYING 'DMKIOS S12174DK D1'
APPLYING 'DMKIOS S12470DK D1'
APPLYING 'DMKIOS S12127DK D1'
APPLYING 'DMKIOS S12629DK D1'
APPLYING 'DMKIOS S12947DK D1'
APPLYING 'DMKIOS S12887DK D1'
APPLYING 'DMKIOS S12984DK D1'
APPLYING 'DMKIOS S13061DK D1'
APPLYING 'DMKIOS S13169DK D1'
APPLYING 'DMKIOS S13206DK D1'
APPLYING 'DMKIOS S12963DK D1'
APPLYING 'DMKIOS S13286DK C1'
APPLYING 'DMKIOS S12941DK C1'
APPLYING 'DMKIOS S13443DK C1'
APPLYING 'DMKIOS S13318DK A1'
FILE 'DKLCLMAC MACLIB' NOT FOUND.
ASMBLING DMKIOS

ASSEMBLER (XF) DONE
NO STATEMENTS FLAGGED IN THIS ASSEMBLY
FILE 'DMKIOS TEXT A1' NOT FOUND.
DMKIOS TXTPTFS CREATED
R;
```

VMFASM will write the new textfile onto your A-disk and will name it DMKIOS TXTPTFS (because the DMKSPLCL control file specified "PTFS" as the update level identifier for the highest-level update applied to DMKIOS, the one listed in your DMKIOS AUXPTFS auxfile.) When you build your new CP nucleus later, you will again use the DMKSPLCL control file, which will tell the loader to pick up a TXTPTFS file in preference to a TEXT file.

I should warn you that VMFASM quietly erases any TEXT textfile you may have on your A-disk for the module you are assembling, even when the new textfile name is going to be TXTPTFS or TXTLCL or TXTTST or whatever. That is what that "DMKIOS TEXT A1 NOT FOUND" message is about; it couldn't find a TEXT textfile to erase this time. This is done to protect naive users from leaving around obsolete textfiles which they might accidentally incorporate into a nucleus. But, it can be a really sneaky problem for the user who is sophisticated enough to want to have more than one version of a textfile. Once you understand what's going on with control files, etc., you should fix VMFASM not to do this to you.* Another danger from VMFASM is that it uses a file named 'modulename cntrlname' as a work file, so if, in this case, you had a file named DMKIOS DMKSPLCL, it would be erased.

You have been worrying about that dump without page zero in it, so you call the Support Center. They haven't done anything about it yet, so you take a look at the code yourself. You see that it would be trivial to fix DMKDMP always to dump page zero, so you produce a temporary local fix to use until you hear from IBM. Because you are expecting this local fix to be replaced by an official fix from IBM, you list the update in an AUXPTFS file, rather than in an AUXLCL file. You name it with its five-digit APAR number, "nnnnn", if it has been assigned one. Some people make a practice of naming local fixes in the form "LnnnnnDK", rather than "SnnnnnDK", to make it obvious that it is a local fix. I find it easier myself to use the name the fix will have when it comes from IBM, but I always put comments into both the fix and the auxfile indicating that this is strictly a local fix. If, as in this case, you have not been given an APAR number yet, use your incident number to name the temporary fix:

FILE: DMKDMP X0328 A1

```
 ./ * TEMPORARY LOCAL FIX FOR PROBLEM OF PAGE ZERO NOT BEING       X0328
 ./ * DUMPED WHEN IT IS ON THE FREELIST                            X0328
 ./ *                                                              X0328
 ./  R  07110000 $ 07110100 100                                    X0328
          LTR   R7,R3              GET THIS PAGE NUMBER.           X0328
          BZ    NXTPAG1            WE ALWAYS WANT PAGE 0 DUMPED.   X0328
```

FILE: DMKDMP AUXPTFS A1

```
 *$*$*$*$*$*$*$*$*$*$*$*$*$*$*$ AUXPTFS $*$*$*$*$*$*$*$*$*$*$*$*$*$*$*$*
 X0328 - 06/23/81 - TEMPORARY LOCAL FIX (DUMP PAGE 0 REGARDLESS) - MWV
```

You reassemble DMKDMP with VMFASM and you have the file DMKDMP TXTPTFS on your A-disk ready to be incorporated into your new nucleus. You rebuild your CP nucleus, using VMFLOAD and the DMKSPLCL control file, and you now have a CP nucleus which differs from the original one by having DMKVMC resident, VM14782 removed, VM13318 applied, and your local fix for incident X0328 applied.

---

* The fix is to comment out two lines from the end of the EXEC:

```
     &IF &TEXT EQ TEXT &SKIP 1
     ERASE &NAME TEXT A1
```

IV. Installing Preventive Service for CP

Things go along more smoothly now, but you have decided that you have been encountering too many known problems. Corrective service is getting to be a drag, so you decide to do some preventive service instead, by installing that VM PUT that came in the mail last week.

A. Ordering the PUT Bucket

The first thing you should do is phone the Support Center and ask Level One to send you a copy of the "error bucket" for this PUT. If you've just come into VM from VS, you will be dismayed to learn that you can't get VM buckets through DATALINK; in fact, there is no such thing as a machine-readable bucket for VM. Let me add that the VM old-timers do not consider this a desirable state of affairs, either. The buckets generally take about a week to arrive, so you need to plan ahead.

B. The Paper Documentation

When you get the PUT in the mail, it is supposed to have a paper memo with it. Find this and read it. Sometimes it tells you about important changes either to the system or to the service installation process.

C. New Service Minidisks

You need a new A-disk for MAINT. If you have a 291 disk to use as an alternate A-disk, then rename it to 191 and rename your old 191 to 291. If you are using the old-user service disk philosophy, you will also use a new disk for the contents of the new PUT. Just flip-flop the 194 and 294 disks from the recommended configuration. If you are using the IBM-recommended service disk layout, you will be loading the new service on top of the base and the earlier service, so you will need only the new A-disk.

D. The PUT Memoranda

For each product on the PUT, there is a "Memo to Users" file. You should always read the memoranda before you apply the service. They may make for rather boring reading, but they tell you about changes to the product which may impact you, so time spent reading them is generally rewarded. The memoranda are on the second file of the tape and have a filetype of MEMO. These commands will load all the memoranda and print the one for SP:

```
attach cuu to maint as 181
tape fsf
vmfplc2 load * memo
print 5664167 memo ( cc
```

Of course, you may prefer simply to view the memoranda on your terminal, rather than printing them.

E. "Mapping" the PUT

The PUT is in the form of one or two physical tapes containing several logical service tapes, one for each PUT-supported product in your customer profile. The first thing you need to do is determine the order of these logical tapes on the physical tape. Even VMSERV has to "map" the tape so that it knows the order of the logical tapes. There is an external label on the tape which will tell you what is where on the tape, but, of course, that's on the tape and the tape is mounted on a tape drive. If you want to load the service yourself, you can copy the external label before you mount the tape or you can map the tape yourself or you can have VMSERV do it for you. You can map the tape yourself with these commands:

```
tape rewind
vmfplc2 scan ( eot disk date
```

This creates a disk file called TAPE MAP which lists the contents of the tape. This file is not very concise, however, so you might find the map that VMSERV builds to be more useful. To map the tape with VMSERV, you do this:

```
attach cuu to maint as 181
vmfplc2 load                  (to load the VMSERV EXEC onto your 191 disk)
access 191 c
vmserv
```

Strange as it seems, you absolutely must run the VMSERV EXEC from a disk that is accessed as mode C. This disk must NOT have a virtual address of 194. VMSERV will insist upon putting some little files on your 194 disk, so if you don't want your 194 written on, detach it and get a 194 TDISK before invoking VMSERV. VMSERV will also print PUT DOCUMENT (the VMSERV documentation) and all the memoranda, if you reply "yes" when it asks you whether you want this done. (Unfortunately, VMSERV insists upon printing the documentation in pieces, rather than as one listing, unless you remember to spool your printer continuous before invoking it.) The map that VMSERV builds for you on your 191 looks something like this:

FILE: SERVICE DISKMAP A1

```
+-------------------------------------------------------------------+
|  * *                                                              |
|  * * VM SYSTEM PUT 8108                                           |
|  * * TAPE LAYOUT AND SERVICE STATUS                               |
|  * *                                                              |
|  * *                                                              |
|  * *     RELATIVE     PROGRAM  NUMBER   FIRST  SERVICE    COREQ   |
|  * *    TAPE - POS    NUMBER   FILES    FILE     LEVEL    FLAG    |
|  * *                                                              |
|          01     01    5749010    37      003    062338    ---     |
|          01     02    5664167    22      040    010923    ---     |
|          01     03    5748XXC    04      062    010405    ---     |
|          01     04    5748XXB    04      066    010505    ---     |
|          01     05    5748RC1    07      070    010408    ---     |
|          01     06    5748AP1    02      077    030203    ---     |
|          01     07    5748FO3    02      079    010103    ---     |
|          01     08    5734LM4    02      081    C40203    COREQ   |
|          01     09    5734LM5    02      083    C40203    COREQ   |
|          01     10    5734PL1    02      085    C40203    COREQ   |
|  * *                                                              |
|  * * END OF RELATIVE TAPE 01                                      |
|  * *                                                              |
+-------------------------------------------------------------------+
```

This can be very useful, especially if you've memorized all your product numbers. I haven't done so, but I know that the logical tape with the most files in it is vanilla VM Release 6, that the other one with a lot of files must be SP, and that the one with the letters "RC" in its product number is PASSTHRU. The other products, whatever they may be, are installed by someone else, so I don't have to worry about the fact that I don't know what they are.

Some of my purist friends will object, but to show how open-minded I am, I am going to recommend that you use VMSERV to map the tape and to print the documentation.

F. Loading the Service from the PUT

Now you are ready to load the service for SP CP. If you look at your VMFPLC2 TAPE MAP or at VMSERV's SERVICE DISKMAP, you will see that the service for SP (5664-167) is in a logical tape containing twenty-two files, beginning (in my example) with file 40 of the PUT. The number and content of these files changes from time to time, but if you examine the SP "Memo to Users" or your VMFPLC2 TAPE MAP, you will see that the layout of the files on this logical tape is something like this:

1. SP installation EXEC
2. CP auxfiles
3. CP updates (PTFs)
4. CP macro auxfiles
5. CP macro updates
6. New CP source
7. CP maclibs
8. CP textfiles
9. CP loadlist EXECs
10. CMS auxfiles
11. CMS updates
12. CMS macro auxfiles
13. CMS macro updates
14. New CMS source
15. CMS maclibs
16. CMS textfiles
17. Standalone IPL decks
18. LOADER and service EXECs
19. CMS module files
20. HELP files and XEDIT EXECs
21. EREP txtlibs
22. IOCP

The CP service is going to go on your 194 disk, if you are using our example disk layout, so make sure that you are using your real 194 and not that TDISK. The commands to load the CP service are simply:

```
access 194 c
tape rewind
tape fsf 2                  <== skip PUT junk
tape fsf 37                 <== skip VM Release 6 files
tape fsf 1                  <== skip SP installation EXEC
vmfplc2 load * * c ( eof 8
```

In other words, all eight files of CP service from the PUT (logical tape files 2 through 9) go onto one minidisk at address 194.

If you are using the IBM disk layout, the same process is:

```
access 194 a
access 294 b
access 394 c
tape rewind
tape fsf 40
vmfplc2 load * * b ( eof 4
vmfplc2 load * * c
vmfplc2 load * * a ( eof 3
```

Or, of course, in either case, you could use VMSERV to load the service. It is usually fewer keystrokes to do it "by hand", however.

That's all you have to do. The service is loaded. At this point, again access your service disks read-only to avoid inadvertently changing them.

G. Carrying Forward Old Corrective Service

The next step in installing a PUT is to check whether you have any old corrective service which needs to be carried forward. You will probably find that some of the fixes you had to apply on top of the old PUT are included in this new PUT and that some of them are not. You still need those that are not in the new PUT, so you must apply them again. If you had removed any bad fixes from the old PUT, you may find that the fixes for the bad fixes are on the new PUT, in which case you should no longer be removing the bad fixes. The fixes for the bad fixes will do that for you and probably won't assemble unless the bad fixes are applied first.

The fixes you had applied on top of your old PUT are on your old A-disk, which is now called 291. Access that at mode F and take a look at it:

```
 listfile * * f

 CPLOAD   AUXPTFS  F1
 CPLOAD   EXEC     F1
 CPLOAD   S12989DK F1
 DKPTFMAC EXEC     F1
 DKPTFMAC MACLIB   F1
 DMKDMP   AUXPTFS  F1
 DMKDMP   TXTPTFS  F1
 DMKDMP   X0328    F1
 DMKIOS   AUXPTFS  F1
 DMKIOS   S13318DK F1
 DMKIOS   TXTPTFS  F1
 DMKMSG   AUXSP11  F1
 DMKMSG   TEXT     F1
 DMKRIO   ASSEMBLE F1
 DMKRIO   TEXT     F1
 DMKSNT   ASSEMBLE F1
 DMKSNT   TEXT     F1
 DMKSPLCL CNTRL    F1
 DMKSYS   ASSEMBLE F1
 DMKSYS   TEXT     F1
 IOBLOKS  AUXPTFS  F1
 IOBLOKS  S13318DK F1
 R;
```

You must examine each of those files to decide whether you need to copy it to your new A-disk.

Control files almost never change except with a new release, and the "Memo to Users" will tell you if they do. So, first thing, copy over your control file:

```
 copyfile dmksplcl cntrl f = = a ( olddate
```

It is worth mentioning here that you should make a habit of using the OLDDATE option of COPYFILE; it's a very simple way to "leave tracks". You should also make a habit of using the REPLACE option as seldom as possible and of always using the TYPE option when you do. At least you'll know then that you've just replaced something that you really wanted to keep.

I would like to suggest that you not copy any of the textfiles or maclibs. It is safer to rebuild them with new VMFASMs and VMFMACs. Start with your macro and copy section changes. You have only one, in this case, that fix that hit IOBLOKS, VM13318. If LISTFILE shows no S13318DK files on your service disks, then you need to carry this fix forward and rebuild your PTF maclib:

```
 copyfile iobloks * f = = a ( olddate type
 COPY 'IOBLOKS AUXPTFS F1' to 'IOBLOKS AUXPTFS A1' (NEW FILE).
 COPY 'IOBLOKS S13318DK F1' to 'IOBLOKS S13318DK A1' (NEW FILE).

 copyfile dkptfmac exec f = = a ( olddate

 vmfmac dkptfmac dmksplcl
```

This will incorporate any new fixes to IOBLOKS which are on the new PUT.

Next, copy over the source for your sysgen modules and reassemble them:

```
 copyfile dmkrio assemble f = = a ( olddate
 copyfile dmksnt assemble f = = a ( olddate
 copyfile dmksys assemble f = = a ( olddate
 vmfasm dmkrio dmksplcl
 vmfasm dmksnt dmksplcl
 vmfasm dmksys dmksplcl
```

The documentation with the PUT is supposed to tell you if these need reassembly, but unless you are very cycle-constrained, it is better to be safe and reassemble them. I can remember a couple of times when they didn't tell us to reassemble DMKSYS when they should have. The resulting system failures were remarkably obscure.

Then, work your way through the rest of the fixes you had applied before to see if you still need them. Of course, you still need the rest of VM13318, the part that applies to DMKIOS:

```
 copyfile dmkios s13318dk f = = a ( olddate
 copyfile dmkios auxptfs f = = a ( olddate
 vmfasm dmkios dmksplcl
```

Again, this will pick up any new fixes to DMKIOS from the new PUT.

Looking at the new DMKMSG AUXSP11 file that was loaded up from the PUT, you see that VM14782 is on the PUT. That is the fix for the PE against VM12684, so you no longer need to remove 12684. If you still needed to remove it, though, you would not just copy DMKMSG AUXSP11 from your old A-disk. The new DMKMSG AUXSP11 that you just loaded up from the PUT very likely would have some additional fixes in it, even if it did not have VM14782. Therefore, the thing to do would be to copy that auxfile from the new PUT disk to your new A-disk and again comment out VM12684, as before.

LISTFILE shows you that there is a file DMKVMC S12989DK that was loaded up from the PUT, so you no longer need to apply the circumvention for APAR VM12989, which, you will recall, was to make DMKVMC resident. If you did still need to apply the circumvention, you would copy your CPLOAD S12989DK and CPLOAD AUXPTFS files from F to A, check that the update would fit properly in the new version of CPLOAD EXEC from the PUT (which might be different from earlier versions), and update CPLOAD EXEC from the new PUT disk onto your new A-disk, exactly as before.

That takes care of everything on the old A-disk but X0328. Just as you are about to carry forward your temporary local fix for X0328, your mail is delivered, and in it you find a letter from IBM informing you that the problem has been closed as a suggestion. So, either you discard your fix and wait for IBM to implement your suggestion, or you convert your fix into a local modification. While I don't want to lead you into temptation, I would suggest the latter, if only to give me a chance to illustrate local mods.

If it is going to be a local mod, it should be listed in an AUXLCL auxfile, rather than in an AUXPTFS. And, I would suggest naming it something other than X0328, say, "DMPZERO0". There are very many schools of thought on naming local mods. Most of the old users seem to have developed elaborate numbering schemes for their mods. I prefer something mnemonic, with a single digit at the end to indicate the level of the mod. Whatever you name your mod, you should, as before, mark it with an update identifier:

FILE: DMKDMP DMPZERO0 A1

```
 ./ * THIS MODIFICATION CAUSES PAGE ZERO TO BE INCLUDED IN      DMPZERO0
 ./ * THE DUMP DATASET EVEN IF IT IS ON THE FREELIST.           DMPZERO0
 ./ *                                                          DMPZERO0
 ./  R  07110000 $ 07110100 100                                DMPZERO0
          LTR   R7,R3               GET THIS PAGE NUMBER.      DMPZERO0
          BZ    NXTPAG1             ALWAYS WANT PAGE 0 DUMPED. DMPZERO0
```

FILE: DMKDMP AUXLCL A1

```
 *$*$*$*$*$*$*$*$*$*$*$*$*$*$*$ AUXLCL $*$*$*$*$*$*$*$*$*$*$*$*$*$*$*$
 DMPZERO0 - 06/23/81 - DUMP PAGE ZERO REGARDLESS - MWV
```

You use VMFASM to incorporate your mod and reassemble DMKDMP. This produces a file called DMKDMP TXTLCL, which the loader will pick up in preference to any files named DMKDMP TEXT or DMKDMP TXTPTFS when you use the DMKSPLCL control file.

With that, you have worked your way through all the files on your old A-disk, and your new A-disk looks like this:

```
 listfile * * a

 DKPTFMAC EXEC     A1
 DKPTFMAC MACLIB   A1
 DMKDMP   AUXLCL   A1
 DMKDMP   DMPZERO0 A1
 DMKDMP   TXTLCL   A1
 DMKIOS   AUXPTFS  A1
 DMKIOS   S13318DK A1
 DMKIOS   TXTPTFS  A1
 DMKRIO   ASSEMBLE A1
 DMKRIO   TEXT     A1
 DMKSNT   ASSEMBLE A1
 DMKSNT   TEXT     A1
 DMKSPLCL CNTRL    A1
 DMKSYS   ASSEMBLE A1
 DMKSYS   TEXT     A1
 IOBLOKS  AUXPTFS  A1
 IOBLOKS  S13318DK A1
 R;
```

With as few files as we had in this example, this procedure is done easily enough by inspection. If there were substantially more fixes, however, you would need a better mechanism to make sure you really processed them all. Many installations have developed EXECs to perform this operation for them. IBM provides no tool for this. I had an EXEC myself for a while, but ultimately I decided that I really wanted to look at what was happening in this part of the process, so I no longer use that EXEC. Now, I copy all of the updates and auxfiles from my old 191 to the new 191 and then just use a LISTFILE EXEC to step through the auxfiles, erasing auxfiles and updates which are no longer needed and making sure that the AUXSP11 files that I still need on my A-disk are current. When that is done, I have only to reassemble all the modules which still have auxfiles on my A-disk, again using a LISTFILE EXEC to drive the process and assure that nobody gets left out. This clearly is a matter of taste, though. Do it whatever way is best for you, as long as all the fixes you need get carried forward correctly.

H. Taking the Actions Described in the PUT Error Bucket

Next you should take the corrective measures recommended in the PUT error bucket. The bucket will tell you about such things as any bad PTFs known to be on the PUT, any changes to program products known to be required because of changes to CP or CMS in this PUT, any PUT packaging errors that have been reported, and other information that you need to have before you put this PUT into production. When the bucket comes, you should read through it carefully, taking the actions it recommends, and marking them off as you go. The buckets tend to be huge and ugly and formidable-looking, but they are important, so you need to learn to understand them. After reading a few buckets, you will come to recognize the unimportant parts and be able to skip over them quickly to get to the useful parts.

The bucket is divided into a number of subsets, one for each component, CP, CMS, DIRMAINT, RSCS, etc. Each subset contains three sections which may be of interest -- Documentation Changes, General Information, and Service Recommendations.

The Documentation Changes sections seldom have anything in them. When they do, they mainly tell you about misinformation in the Memoranda to Users from the PUT. Read these comments and note them.

The items in the General Information sections are almost all things you need to know. Most of the items there are directions for fixing up packaging errors on the PUT. The descriptions of the actions you need to take are generally reliable, although skimpy. They assume you know how to do the sorts of things I've been talking about. They may tell you, for example, to remove a certain bad fix, or to edit an auxfile and reassemble some module to pick up a fix they left out, or to generate such-and-such a CMS module which they forgot to regenerate after they updated the textfiles.

Other items in General Information may tell you that one of the sysgen modules (DMKRIO, DMKSNT, or DMKSYS) needs to be reassembled because of a change to a macro on this PUT. There may also be notes telling you about new permanent restrictions or warning you to apply some important corrective fix. The information in this section is generally reliable. When it tells you to do something, you should do it. It will usually be reasonably clear whether the item applies to you or not.

One particular class of General Information items you should be aware of is notes telling you to apply a certain fix to some component, say, DIRMAINT, when you apply the PUT service to CP or CMS. They do not usually tell us about these "co-requisites", but when they do, they mean it. Take careful note of such advice.

The Service Recommendations sections are the part of the bucket that you will find hardest to digest. These sections are essentially just dumps of APARs from RETAIN. This is a typical example:

```
+-------------------------------------------------------------------+
| APAR NUMBER = VM13499                LAST UPDATE = 81/09/23       |
| PTF IN ERROR = UV03973               PIN = YES                    |
| CURRENT APAR STATUS = CLOSED         CLOSING CODE = PER           |
| ORIGINAL APAR NUMBER =               SECURITY/INTEGRITY = NO      |
| SERVICE NUMBER = X401302-                                         |
| REPORTED COMPONENT = 5749DMK00       RD62       5749 VM/370 CP   |
| FIXED    COMPONENT = 5749DMK00                  5749 VM/370 CP   |
| FAILING MODULE = DMKFRE                                           |
| PROBLEM ABSTRACT:                                                 |
| DMKCPU USES WRONG REG AND WRONG INSTRUCTIONS WERE IN DMKFRE       |
| REPORTED SCP RELEASE:   D62                                       |
| ERROR DESCRIPTION:                                                |
| APPLICABLE PTFS: PE03973-T8104, PE03974-T8104                     |
| VM12596 DMKCPU FIX USES REG4 TO STORE THE CPEXBLOK ADDRESS INTO   |
| THE FREE STORAGE BACK POCKET DMKFREAP 'CPUFREAP'.  REG10 READ     |
| ADDRESS IS THE REGISTER LOADED WITH THE ADDRESS OF 'CPUFREAP',    |
| AND IS THEREFORE THE CORRECT REGISTER TO USE FOR THE STORE.       |
| ALSO, THE DMKFRE FIXES FOR R060, RD61, RD62 ARE INCORRECT.  THE   |
| WRONG INSTRUCTION WAS REPLACED.                                   |
| TEMPORARY FIX:                                                    |
| PRE-REQ APAR IS VM12596                                           |
| FIX FOR WRONG REGISTER USAGE IN DMKCPU:                           |
| FILE: DMKCPU R13499DK FOR R060                                    |
| ./ R 426500 $ 426600                                              |
|          ST    R1,4(,R10)   PUT CPEX POINTER IN DMKFREAP  @VA13499 |
| **** END OF REL6 FIX****                                          |
| FILE: DMKCPU S13499DK FOR RD71                                    |
| ./ R 4317000 $ 4318000                                            |
|          ST    R1,4(,R10)   PUT CPEX POINTER IN DMKFREAP  @VA13499 |
| **** END OF VM/SP FIX ****                                        |
| FIX FOR DMKFRE FOR R060:                                          |
| FILE: DMKFRE R13499DK                                             |
| ./ R 758100 $ 758100                                              |
|          ICM   R2,B'0111',DMKFREAP+1  ADDR OF BACKPOCKET  @VA13499 |
| ./ R 762000 $ 762100                                              |
|          L     R1,DMKFREAP+4  GET ADDR OF CPEX FROM FREAP @VA13499 |
| **** END OF R060 FIX ****                                         |
| PROBLEM SUMMARY:                                                  |
| VM12596 - DMKCPU FIX USES REG4 TO STORE THE 'CPEXBLOK' ADDRESS    |
| INTO DMKFREAP INSTEAD OF REG10. THE DMKFRE FIX FOR R060 IS ALSO   |
| INCORRECT: THE WRONG INSTRUCTION WAS REPLACED.                    |
| AFFECTED USERS: AP AND MP USERS ONLY                              |
| RECOMMENDATION: APPLY APAR TEMPORARY FIX                          |
| PROBLEM CONCLUSION:                                               |
| DMKCPU WILL BE CHANGED FOR R060 AND RD71. DMKFRE WILL BE          |
| CHANGED FOR R060.                                                 |
| MODULES/MACROS:   DMKFRE   DMKCPU                                 |
| SRLS:     NONE                                                    |
| CIRCUMVENTION:                                                    |
| APPLICABLE COMPONENT LEVEL/SU:                                    |
| ENV=060         PS=Y  PTF=UV04522 OPEN    AVAIL=NO   VOLID=       |
| ENV=D71         PS=Y  PTF=UV04523 OPEN    AVAIL=NO   VOLID=       |
+-------------------------------------------------------------------+
```

You have got to work your way through these APARs, deciding what to do about each one. There are some clues. Each of them has an "AFFECTED USERS" line (toward the end and very hard to spot). In this case, it says "AP AND MP USERS ONLY". This field frequently says "ALL" when the fix really doesn't apply to all installations. In fact, when this particular APAR was first entered into the bucket, it said that all users were affected, even though the fix is to DMKCPU, which is not even listed in the UP loadlists. But, if the "AFFECTED USERS" field says "MP-ONLY" and you have a UP, or if it says "FBA-ONLY" and you have only CKD devices, then you are safe in skipping that item. In other cases, the problem description will make it clear that you are not impacted. For example, if it is a fix for a problem in IUCV and you know that you are not using IUCV, then you can skip that one, too. You should be aware that some of the APARs in the Service Recommendations sections are marked "HIPER", which is supposed to mean that this is a high-impact problem. This is a new designation, and so far it has not been terribly accurate, but it does give you some guidance.

When you decide you have to do something about an APAR in the Service Recommendations section, you may have several options. Look at the "PTF IN ERROR" line, the "RECOMMENDATION" line, and the "CIRCUMVENTION" line. There is very seldom anything given as a circumvention, but if there is, it is likely to be something relatively safe and easy to do, such as making a module resident. If there is no circumvention, you may have the choice between applying this fix and removing some bad fix that this one is supposed to correct, the one mentioned as the "PTF IN ERROR".

You should keep in mind that the fixes in the bucket are for the most part even less well-tested than the fixes on the PUT, so you are asking for trouble if you just blindly apply them all, even if that is what the "RECOMMENDATION" fields tell you to do. If this APAR is a fix for a "PTF in Error" ("PE"), you may be better off just pulling the bad PTF, rather than installing this APAR. To pull off the PE, the first thing you have to do is figure out the APAR number that corresponds to the PTF number given in the "PTF IN ERROR" line. All VM fixes have both an APAR number ("VMnnnnn") and a PTF number ("UVmmmmm"). VM fixes are named according to their APAR number, rather than their PTF number, and VM APARs are never superseded by PTFs, as VS PTFs are. Therefore, in VM, PTF numbers convey no information to the user, but IBM puts the PTF numbers in the VM buckets, rather than the APAR numbers, because what's good for MVS is good for VM.

Given the PTF number for the PE, here is how you find out its APAR number, so you know its filetype, so you can remove it. If you are lucky, the "PROBLEM DESCRIPTION" field will reference the bad fix by its APAR number, as it does in this case, VM12596. Or, the bad fix may be listed as a "PRE-REQ APAR", as it is here. If not, you could just call Level One and ask them what the APAR number is for PTF UV03973. Or, if you have INFO/DATA, you could look up the PTF number there; the command is this case would be "S E UV03973".

If you don't have INFO and you don't want to call the Support Center, you can try a little detective work. A reasonable assumption is that the PE probably hit at least some of the modules that the fix for the PE hits. You have the fix for the PE sitting there in front of you. So, look at the AUXSP and AUXSP11 files for the modules the fix applies to; you will probably find the PTF number of the PE listed in one of them. The APAR number will be on the same line, because it is included in the name of the update, S12596DK. Now that you know the APAR number of the bad fix, you can find out what the original problem was that it was trying to fix. Look up the APAR number in Early Warnings and read the problem description. (Of course, if you phone the Support Center to find out the APAR number, you can also ask them to read you the problem description. Similarly, if you look up the PTF number in INFO, that will also give you the problem description.)

If you decide that the original problem is not one that you have had, it may well be that you will get more stability from removing the bad fix than you will get from applying the fix for the bad fix. If you decide to remove the bad fix, you go through the same process that we went through to remove the bad fix to DMKMSG earlier. You use LISTFILE to find all the macros and modules which are hit by the PE. You copy the AUXSP11 (or AUXSP) auxfiles for those macros and modules from the service disk to your A-disk, and you edit the A-disk copies of the auxfiles to comment out the bad fix. If the bad fix hits any macros, you add the macros to your DKPTFMAC MACLIB by adding entries for them to DKPTFMAC EXEC and issuing the VMFMAC command to rebuild the maclib. (You could update the macro by hand and then add the updated macro to DKPTFMAC with the MACLIB ADD command, but your PTF macro libraries will generally be so small that it will be very inexpensive to rebuild them from scratch when you need to make any change in them. And a problem with using MACLIB ADD to add a macro to a maclib is that it is so easy then to forget to add the name of that macro to the EXEC for that maclib. This means that the next time you rebuild that maclib from scratch, you will lose the new macro and the service that was on it.) Once you have the auxfiles and the maclib in order, you use VMFASM to reassemble all the modules that were hit by the PE.

If, on the other hand, you decide that you really should apply the fix given in the bucket, you use the same procedure that we used earlier in applying the fix for the hungusers problem. First, key in the update files from the bucket. Then, create (or update) an AUXPTFS auxfile for each macro and module hit by the fix. If any macros are involved, add them to DKPTFMAC, in the usual way. Then use VMFASM to assemble the modules and create TXTPTFS files on your A-disk.

That is all you do with the APARs in the Service Recommendations sections of the bucket -- skip them, apply them, or pull the PEs they are supposed to fix. There is another bucket you need to check, however, the one from VMSHARE. There are generally a series of files on VMSHARE which discuss current problems with the system. For SP, these have been named PROB 1SP00, PROB 1SP01, etc., to correspond to the various service levels. Read all these buckets, since they are not cumulative. You may not need to apply all the service that is recommended there, but many of us have found that that is the safest course for us.

I. Carrying Forward Your Local Mods

We did not have any local mods on the old 191 disk in this example. If we had, they would now need to be copied to the new 191. The process here is really just the same as it is with locally-applied service. Copy over the AUXLCL auxfiles and the local updates. Rebuild DKLCLMAC MACLIB, using VMFMAC with the DMKSPLCL control file. Then reassemble the modified modules, using VMFASM and the DMKSPLCL control file.

With enough effort, you could probably figure out that some of the old TXTLCL files could just be copied forward, because those modules had not received any new service. I would advise against this approach, though, because the checking required is really substantial. You would have to be sure that the macros used by those modules also had not received service that would require a reassembly. And, if you were to miss just one module that really needed reassembly, you would have built yourself a very obscure problem to debug.

In carrying forward your local mods, you need to check, of course, that they still fit in IBM's code after the new service has been applied. It will be well worth your time to read through the APAR descriptions in the Memo to Users, looking for fixes which sound as if they might hit areas you have modified. Fortunately, you can count on UPDATE and the assembler to find most of the conflicts between your mods and the new service for you. So, the first step is the VMFASMs I suggested above. Check very carefully for both UPDATE sequence error messages and assembly error messages. When you have all of those fixed up, you really should examine all the mods in the new listings to make sure they still make sense, but this is so tedious that I suspect that many of us skip that step and just rely on functional testing to make sure our mods are all still working. Certainly, it is worth spending some time building yourself a testing EXEC that exercises all your mods. You may also want to consider modifying the VMFASM EXEC so that it will not go on and do the assembly when there is an UPDATE error. This will make UPDATE errors much more conspicuous. Many of the old-timers are of the opinion that this is how VMFASM should work. So far, we've not been able to convince IBM of that, however.

J. Testing a New Version of CP

Having done all this, you are ready to test your new version of CP.

I'm pretty well convinced that the primary reason that VM still lives is the fierce loyalty of the VM system programmers and that the primary reason for their loyalty is that VM means never having to take standalone test time at 6 A.M. on Saturday.

```
+-------------------------------------------------------+
|                    RULE NUMBER FIVE                   |
+-------------------------------------------------------+
|  VM SYSTEM PROGRAMMERS DO IT VIRTUALLY ALL THE TIME   |
+-------------------------------------------------------+
```

It is my impression that most experienced VM installations never schedule standalone test time for CP. A new version of CP can be tested quite thoroughly running in a virtual machine. It will take you a little effort to learn how to do this effectively, but the effort will be repaid many times over. A good place to start is by reading the chapter on running VM under VM in GC19-6212, "Operating Systems in a Virtual Machine".

If you are new to VM system programming, you will also need to spend some time learning to read dumps interactively using the DUMPSCAN facility of IPCS or Amdahl's ANALYZE (if you are lucky enough to be able to get hold of it). DUMPSCAN is a frustratingly primitive facility, but, even so, it can greatly enhance your productivity. Spend a little time learning to read dumps interactively, and you'll soon be refusing to look at paper dumps.

K. Putting a New Version of CP Into Production

Finally, you are ready to build and install a production CP nucleus at the new PUT level. The mechanisms you use are exactly the same as the ones you use in building and installing a new nucleus to incorporate a single fix.

Even after you have this PUT in production, you should order a new copy of the bucket every once in a while and go through it to see if any more problems have been discovered. How often you need to do this depends on how stable your system is. In the early days of SP, when things were very unstable, I checked the bucket every week. Fortunately, at that time, the bucket was in a format which allowed Level One to tell us easily whether there had been any important changes since we last got a bucket. It is no longer possible for us to phone Tampa or Chicago and just ask what is new in the "hotlist", but I hope this situation will be remedied someday.

V. Converting to a New Release of CP

Once you have a VM system up and running, installing a new release of VM is exactly the same process as installing a new PUT, so don't let them trick you into thinking that you ever have to use that nasty starter system again. The steps we have just gone through to install a new PUT are the same steps you need to take to install a new release. Essentially, all you do is load up the tape, do whatever the bucket tells you to do to fix it up, apply any old corrective service you still need, and then apply your local mods.

A. Loading Up the Base Tape (and Possibly a Service Tape)

I cannot give you the exact commands you will need for loading up the new release, because the tape formats tend to be different with each new release. The documentation with the tapes may be sufficient to enable you to find the files you need. If it is not, use VMFPLC2 (or whatever they call it next time) to scan the tapes and show you what is there. For CP, you need to find a file with source and macros and a file with textfiles. (When we went to Release 6, the only place on the distribution tapes that we could find the textfiles was on a minidisk in the starter system! Apparently, they thought that people actually use the starter system to go to a new release. We can hope that they will remember the shrieks of pain and outrage and not do that to us again for a while.) The control files, loadlists, and maclibs for CP will also be somewhere on the tapes. Once you have found all these, get yourself a new base minidisk (like 195 in our canonical layout) and load all the CP files up onto it.

New releases tend to be pretty badly back-levelled on service, so, unless you are really desperate to get onto the new release for some reason, wait at least until the first service tape arrives. When it does, get yourself a new PUT minidisk and load the CP service up onto it.

B. Doing What the Bucket Says

Either way, get yourself a bucket and take a look at the subset which describes problems in installing the new release. Get yourself a new A-disk and do whatever fixups the bucket suggests. Then start working your way through the task of carrying forward old corrective service and local mods.

C. Carrying Forward Old Corrective Service

Since even with a service tape the new release may be back-levelled, it would be a good idea for you to make yourself a list of all the corrective service you have had to apply in the past six months. Work your way through the list deciding which of those fixes have been picked up for the new release and which have not. Then phone the Support Center and ask for new versions of all the old fixes which have not yet been picked up. Do not try to update these fixes yourself; the Support Center will generally send you tapes of the updated fixes within a week or so of your asking for them.

D. Carrying Forward Your Local Mods

Moving your mods into the new release is much like moving them into a new PUT. It may be complicated, however, by the base ASSEMBLE files having been resequenced. If that happens, the sequence numbers in your mods will need reworking, even if the logic doesn't. You should be aware of a very useful program on the Waterloo tapes, the extended COMPARE command from Cornell. One of its many functions is to prepare a resequenced update file when it is given an old ASSEMBLE file, an update to that file, and a new, resequenced ASSEMBLE file. I would still be trying to get to SP if it were not for this program.

E. Compatibility Problems

The one real complication you may encounter is release-to-release incompatibilities, most likely in spool files or the directory. We have been telling IBM for a long time that we must be able to go back and forth between releases without doing cold starts and without rebuilding our directories. The user community has traditionally built and distributed the modifications necessary to achieve these goals. In the Release 6 to SP conversion, IBM finally came through with a PTF to allow us not to do cold starts either when installing SP or when backing out of it. They still did not achieve directory compatibility, but the users did, and there is hope that IBM will next time. You can expect to find directions for use of any compatibility PTFs in the install bucket for the new release. Changing the size of, say, the SFBLOK without cold starting is a tricky process, so follow the instructions very carefully.

VI. Advanced Nucleus Theory

A. Building and Delivering a CP Nucleus

To build a CP nucleus, you just spool your punch to yourself and invoke VMFLOAD to punch to your reader a big file containing the CP loader followed by all the textfiles for the nucleus. At some point you deliver the nucleus to your sysres by IPLing that loader file. IPL gives control to the loader; and the loader reads in the rest of the deck, link-edits the whole thing together, and prints the loadmap. It then passes control to DMKSAVNC, which writes the nucleus out onto the sysres volume described in DMKSYS.

There are several different ways to go about accomplishing this delivery of the nucleus:

1. Perhaps the most common delivery method is to move that reader file to a tape and later IPL the loader from that tape into the real machine to write the nucleus to the real sysres:

```
tape rew
filedef input reader
filedef output tap1 ( recfm f lrecl 80 block 80 den 1600
movefile input output
```

Either VMSERV or GENERATE can also be used to create an IPLable nucleus tape.

2. Or, as in our earlier examples, you can just get write access to the real sysres and IPL that reader file in your virtual machine to write the nucleus out onto your real sysres. This is risky enough that I would advise against it, however, until you feel that you really understand what is going on.

3. You can also set up a virtual sysres on a minidisk, giving it the same address and volid as the real sysres and CP-allocating it. You then IPL that reader file in your virtual machine, and the loader writes the new nucleus onto the virtual sysres. After you have tested the new nucleus in your virtual machine and are happy with it, you can use DDR COPY NUC to copy it from your virtual sysres to the real one. A somewhat less gutsy approach is to use DDR DUMP NUC to move the nucleus from the virtual sysres to tape, after which you shut the real system down at some point and install the new nucleus using the RESTORE NUC function of standalone DDR.

4. You can use RSCS to send your reader file to a virtual reader on another real machine and then use one of these three methods to install the nucleus on that machine.

I have used all of these approaches myself and currently use different ones for different systems. I would advise you to start with one of the approaches that doesn't involve writing on the real sysres while the system is up and running. I will be discussing the perils involved in doing that shortly. I do recommend that which ever way you deliver your new nucleus to your real sysres, you first load every new CP nucleus to a virtual sysres and IPL it from there. It's so humiliating to have the operators tell you that you've given them a new system that won't even IPL.

B. Defining A V=R Area: DMKSLC

If you build your nucleus "by hand" or with your own build EXEC and you want to have a V=R area in your system, be sure that you have generated a DMKSLC textfile. (You must also, of course, use a loadlist which lists DMKSLC.) DMKSLC comes right after DMKPSA (page zero) in the CP nucleus. "SLC" stands for "set location counter", which is precisely what DMKSLC does. It moves the location counter from the beginning of page one to the end of the V=R area, which is just a big hole in your CP nucleus. You use the VRSIZE command to generate DMKSLC:

```
vrsize
VIRTUAL=REAL OPTION REQUIRED (YES,NO):
yes
STORAGE SIZE OF VIRT=REAL  (MINIMUM IS 32K):
5m
STORAGE SIZE OF VIRT=REAL  (MINIMUM IS 32K):
5120k
05120K STORAGE SIZE FOR VIRTUAL=REAL
IS THE ABOVE ENTRY CORRECT (YES,NO):
yes
R;

l dmkslc ( d
FILENAME FILETYPE FM FORMAT LRECL       RECS     BLOCKS   DATE     TIME
DMKSLC   TEXT     A1 F         80          3          1  1/16/82 14:24:57
R;
```

Just reply to the questions, and VRSIZE will build a file called DMKSLC TEXT. As you can see, you must tell it the V=R size in K, rather than M -- very unfriendly. You need do this only once, as long as you don't want to change your V=R size, and you may keep around different DMKSLC textfiles for different purposes. For example, I keep two versions of DMKSLC on my MAINT A-disk, DMKSLC TXT3033, which defines a 5-meg V=R for my 3033, and DMKSLC TXTTST, which defines a 2-meg V=R for my test system.

C. Detecting Nucleus Overflow

One of the hazards of building CP nuclei is the problem of "nucleus overflow". There is no check made for whether the nucleus is too big for the DASD space you've set aside for it. In fact, you specify only the starting cylinder or block for your CP nucleus in DMKSYS. The system neither knows nor cares where you intend for the nucleus to end. DMKSAVNC starts writing the nucleus at your starting point and keeps writing until it's done. If it goes too far, whatever comes next on that volume is quietly destroyed.

Early in SP, IBM came out with a very friendly fix to DMKSAV which tells you exactly which cylinders or blocks the nucleus has been written on. You should certainly always check that message. But, since there's a chance that you might overlook it, you should also adopt one of the old-timers' strategies for detecting nucleus overflow.

One good strategy is to lay out your virtual sysres so that the I/O error recording area is immediately after the CP nucleus. Then, when you load your new nucleus onto your virtual sysres, if the nucleus overflows, it will overlay the error recording cylinders. Since CP initialization carefully checks to make sure that the error recording area is properly formatted, you will get the message:

```
 DMKIOG552I FORMATTING ERROR RECORDING AREA
```

when you IPL your virtual sysres under these circumstances. Then you know not to install on your real sysres without first rearranging it a bit.

There is another easy way to detect nucleus overflow. When you IPL the loader reader file in a virtual machine, you must have the sysres volume (whether virtual or real) defined as a minidisk to your virtual machine. If you define that minidisk as ending at the end of the nucleus area, then, if the nucleus overflows, you will draw I/O errors when DMKSAV tries to write beyond the end of the minidisk. Again, you are warned that you have a problem.

D. Backing Your Nucleus Up

The most important part of building a new CP nucleus is backing up the previous nucleus, so that you will be able to back out of the new one.

```
+--------------------+
|  RULE NUMBER SIX   |
+--------------------+
|    BACK IT UP      |
+--------------------+
```

Two of the delivery methods I described just now involved your writing the nucleus to tape. If you do it yourself, then you will know it really got done and you will know what tape the nucleus is on. If you deliver your nucleus direct to the sysres, rather than writing it to tape, make sure that your operators have good system backup procedures and that they really follow them.

```
+----------------------+
|  RULE NUMBER SEVEN   |
+----------------------+
|  BACK IT UP AGAIN    |
+----------------------+
```

Keep more than one nucleus backup tape. Set up your backup procedures to use a rotating pool of several tapes on which to dump new nuclei in turn. (We find ten to be a good number.) And when you really screw things up, you will be happy that you have dumped your MAINT A-disk out onto the backups too, following the nucleus. That way, when you back out to an old nucleus, you can also back your A-disk up to the state it was in at the time you built the good nucleus, if you need to.

```
+----------------------+
|  RULE NUMBER EIGHT   |
+----------------------+
|   DON'T TRUST DDR    |
+----------------------+
```

Sad to say, DDR has been very unreliable ever since we went to SP. PUT 8209 includes forty-six fixes to DDR, most of which represent situations in which somebody's backup tapes were no good. I am not saying not to use DDR -- we really have no choice -- I am just saying not to trust it. Thoroughly test every DDR procedure you establish, to make sure that DDR works with the particular combination of devices and commands that you are using. And don't believe that DDR is working just because it says it is. Restore whatever it is to another volume and compare the restored data with the original data. Also, if part of the procedure calls for using DDR standalone, then test it standalone; IPLing DDR in a virtual machine is not the same as IPLing it in a bare machine. And be VERY suspicious of new service for DDR. DDR is like CPEREP; if you find a version that you like, stick with it, or, at least, keep a copy of it around for use in emergencies.

E. Logging and Numbering Your Systems

A good system log, whether in the form of a CMS file or of an old-fashioned logbook, can be invaluable when things start going wrong. There should be one place where your installation logs the time and date of all system changes, both hardware and software. When you install a new CP nucleus or make any change to CMS, you should log that fact and include a detailed description of your changes, including APAR numbers, etc.

Many installations find it useful to number their CP nuclei. In the case of my own installation, the system number is recorded in the system log, of course, but it is also recorded in the CP nucleus itself to help keep dumps straight. This number is displayed at IPL time, and there is an operator command to display it as well. Our nucleus build EXEC automatically increments this number and reassembles DMKCPE, which is where the number is kept. I have included this very simple modification in an appendix in the handout.

F. Archiving Your Load Maps

Archive your CP (and CMS) loadmaps in some orderly way, so that you can retrieve them for use in looking at dumps and documenting problems. If DASD or tape space for archiving loadmaps is a problem for you, then you can discard the loadmaps for systems that don't produce any dumps or other problems. If you make a practice of numbering your system builds, then name the loadmap disk file accordingly, as NUC175 MAP, say; otherwise, name it according to its build date. Before you put a new CP into production, make sure that you have already sent a copy of the loadmap to your IPCS virtual machine.

The loader creates the loadmap while it is building the nucleus. If you are planning to do the build on a bare machine, first IPL the loader file in a virtual machine to get a copy of the loadmap. (Remember to spool your reader HOLD, so that you don't lose the loader file.) Even if you don't have a virtual sysres to load the new nucleus onto, you can trick the loader into creating a loadmap spool file. By the time it has discovered that it has no sysres on which to write the nucleus, it will already have built the loadmap in your virtual printer. So, just close the printer after it tells you it got an I/O error trying to write out the nucleus.

G. Unresolved References

Part of what the loader does, of course, is resolve all the external references from one nucleus module to another. Ideally, there should be no unresolved references. You can find out whether there are by looking at the end of your nucleus loadmap. If there are any unresolved references, you had better find out why, because there is a good chance that your system is missing function and also a good chance that it will branch to zero now and then. The one time that you neglect to check the unresolved references before putting your nucleus into production will be the one time you manage to build a nucleus with no DMKSNT in it. I speak from experience.

RULE NUMBER NINE
CHECK THE UNRESOLVED REFERENCES

Unfortunately, you may find that your CP nucleus has some references which are supposed to be unresolved. In VM/SP Release 1, you may have three such unresolved references. If you build a system without a V=R area, references to DMKSLC will be unresolved. If you don't specify a NAMENCP macro in DMKSNT (because you don't have a 370X), then you will get an unresolved reference to DMKSNTRN. If you don't specify a NAME3800 macro in DMKSNT (because you don't have a 3800 printer), then you will get an unresolved reference to DMKSNTQN.

Unresolved references can be a real problem, because it is so easy, when you are supposed to have seven of them, to overlook the eighth one that sneaks in -- until it crashes your system, of course. But, suppose you decide that having all these unresolved references is slovenly and dangerous, and that you are going to do a little mod to artificially resolve all the references you normally have unresolved. It sounds good, but if you do it, you are likely to find your system crashing all over the place. This is because our friends the developers have a nasty habit of using the fact that an ADCON is all zeroes as a flag. If you resolve the reference to something other than zero, they decide that you are using that function. You can trick them, however, by resolving such references to zero.

Larry Brenner of Cornell has written a very nice little program called NUCMAP, which is available on the Waterloo tapes. NUCMAP reads a loadmap spool file and creates two small disk files containing the names and addresses of all the modules in that nucleus, sorted alphabetically in one file and numerically in the other. At the same time, NUCMAP displays all the unresolved references on the console. I find that this is a good, quick way of checking for unresolved references; you might too. Incidentally, when you are "fixtesting" a new fix, it is a good idea to check for any new unresolved references introduced by the fix and to complain if there are any.

H. The Small CP Nucleus Option

You may want to consider using the small CP nucleus option. The small CP nucleus is just a CP nucleus built with a loadlist that doesn't have all the modules listed in it. This loadlist is called CPLOADSM EXEC. Specifically, the small nucleus doesn't have MSS support, 3340 support, AP/MP support, remote 3270 support, SNA support, or the SEPP fast-path privop handling for guest SCPs. It can save you quite a bit of memory on small systems. In our example, the command to build a small CP nucleus would be:

```
vmfload cploadsm dmksplcl
```

VMSERV will not let you specify that you want to use the CPLOADSM loadlist to build your nucleus, but GENERATE will. However, neither of them will let you specify your own control file.

Unresolved references get to be a much bigger problem if you use the small CP loadlist, however. Many modules are not there, so you get many unresolved references -- dozens of them. This is a real hazard, so I have included in an appendix to the handout the mod I use to artificially resolve all of the normally unresolved references in my small CP nuclei.

I. Alternate CP Nuclei

Sooner or later, you will build a really awful CP nucleus. It will run just fine when you are testing it in a virtual machine, but on the real machine it will not IPL at all or it will crash as soon as you bring it up, every time you bring it up, or it will go straight into a loop every time you bring it up. When this happens, you will be called to the machine room and told to do something about it, while the operators sneer and crowds of angry users mill about, crying for system programmer blood.

```
+--------------------------+
|      RULE NUMBER TEN     |
+--------------------------+
|  PLAN ON BACKING IT OUT  |
+--------------------------+
```

Certainly, this is a good time to have a sound CP nucleus on a sound tape and to know which tape that is. But with all that shouting going on, it can take a while to find the tape and load it up. It would be much nicer if you had another nucleus online and could just tell the operator to change the load address and IPL. Many of the old users have long since concluded that the better they can back out, the better their raises are, so they have become very good at backing out. That is why one of the most common mods floating around among the old users is the "alternate nucleus" mod for CP. This mod allows one to have CP nuclei scattered about here and there, one per DASD volume, rather than forcing the nucleus to be on the sysres volume (the volume with the warm start, error recording, and checkpoint areas). My smallest system has two nuclei, the current one and a recent one that I liked. My largest system has four nuclei, the current one, the one before that, the one before that, and a spare.

The spare is usually a non-V=R system that I keep around because we can't IPL our V=R system in the IBM memory. Sometimes, though, I put a new system there with a new I/O configuration. If I want to go away for the weekend, for example, and I know that the CEs may not be able to get the new gear they are installing on Saturday morning to fly, I just let the operators know to IPL 843 if the change goes and to IPL 840 if it doesn't. In the early months of SP, I kept a really good old BSEPP nucleus on the spare as insurance. That is the one I backed out to so I could go to SHARE. If I could have only one mod on my system, this is the one I would choose. (The second would be the University of Maine's virtual PER support.) The alternate nucleus mod and some examples of its use are included in an appendix in the handout.

J. The Perils of SHUTDOWN

You may have been surprised earlier to hear me suggest that you might install a new CP nucleus on your real sysres while your system is up and running. On the other hand, you may have been surprised to hear me suggest that this is a risky undertaking. All of IBM's higher-level service tools (IPF, VMSERV, and GENERATE) behave as though it is perfectly safe to install a new CP nucleus while the system is running. But many experienced VM shops regard this as altogether too dangerous even to consider doing it. In my view, there are two circumstances in which it is reasonable to install a new CP nucleus on a live sysres: (1) you don't mind doing cold starts or (2) you actually understand what is going on.

Installing a new nucleus on a live sysres is made possible by the fact that during initialization the nucleus modules are read into memory from the sysres, and then the pageable ones are written out to paging devices. Subsequent references to pageable modules cause them to be brought in from the paging area, not from the sysres. So, you are free to write on the sysres, except that there is one real gotcha -- DMKCKP, the SHUTDOWN module. DMKCKP lives on the sysres. When your system gets shut down, either because the operator issues the SHUTDOWN command or because the system crashes, DMKCKP is loaded into memory from the device from which the system was last IPLed, not from a paging volume. DMKCKP is also loaded in from the sysres when you IPL the sysres to do a shutdown and, of course, that means any time you IPL CP and it finds that address X'370' still says "CPCP". Now suppose that your system is running and you write a new nucleus onto the sysres. When something happens to cause your system to be shut down, it is the new version of DMKCKP that is going to be brought in to shut the old system down, so you had better be sure that the new CKP is compatible with the old system.

There are two major areas in which you may have problems with incompatibility between a new DMKCKP and an old nucleus. The first is the result of changes in the spooling subsystem. One of the main things SHUTDOWN does, of course, is move information about your spool files out to the warm start area, so that you can bring your system back up with a warm start. If the new system expects the SFBLOKs, the SPLINKs, the RECBLOKs, or the ALOCBLOKs to have a different size or shape from that they had in the old system, your warm start area is likely to end up full of garbage. However, IBM is not going to change these things without telling you, and you should avoid changing them yourself and should do it very carefully if you must. The only time IBM is likely to make such a change is going from release to release. When they have to do this, they generally give us a compatibility PTF to change the old system's control blocks to look like those for the new. When you build a nucleus to install such a PTF, do not shut down the previous system with the new DMKCKP. And, in general, never shut down a system with a DMKCKP from another release, whether or not a compatibility PTF has been applied. A cold start is the best thing that is going to happen to you if you try that. Again, I speak from experience. If you have installed a new release and things are going so badly that you are unable to shut down before you back out to the old release, hit stop, clear memory, and then IPL the old nucleus and do a CKPT start.

The other thing which may make a new DMKCKP incompatible with an old nucleus is the fact that CKP must be able to use various control blocks and system service routines. If it can't find them, it won't be able to shut your system down correctly. Of course, the offsets to these things may be different from one nucleus to the next, since the length of various nucleus modules may have been changed by new service or mods. So, the new CKP must be able to find the things it needs in the old nucleus without using VCONs. The way it does this is to use the pointer ARSPPR, which is always at the same location in page zero, to lead it to a list of VCONs in DMKRSP. This list contains all the external references that DMKCKP needs. And since the list is part of the old system, it shows CKP where those items are in the old nucleus that it is trying to shut down. This mechanism works perfectly well until somebody breaks it -- either you or IBM. If you modify DMKCKP, be sure that your mods don't add any EXTRN statements to CKP. If you need a new external reference, add it to the end of that list in RSP. IBM has been known to break this mechanism itself on a number of occasions, so if they ever ask you to fixtest a fix to DMKCKP which introduces an EXTRN statement, protest vigorously. SP, as released, has two such EXTRNs in CKP, one for DMKRIOCN and one for DMKSYSFL. These have caused a lot of systems not to shut down reliably. There is a fix available, VM15048, which adds those EXTRNs to the end of the list in RSP and fixes CKP to refer to them through that list rather than through its own EXTRNs. Caution is required when installing such a fix, however. Assuming that you are operating in a mode of installing your new nuclei on the sysres with the system running, such a fix or mod should be applied over two separate IPLs. You should first build and install a nucleus which adds the items to the end of the list in RSP and then later build and install a nucleus which changes CKP to expect to find the new items in the list.

Another caveat: if you increase your V=R size, don't shut the old system down with the new DMKCKP. During SHUTDOWN, CKP is loaded into memory at X'800' bytes beyond the end of the V=R region specified in the new nucleus. You risk either overlaying CKP on top of service routines that it needs (if you make the V=R region a little bigger) or overlaying it on top of SFBLOKs in free storage (if you make the V=R much larger).

One of the other perils of SHUTDOWN is simply that it is so buggy. In fact, the whole SHUTDOWN/WARM START/CKPT START subsystem is just full of bugs, especially in I/O error recovery. The VM community is subjected to altogether too many cold starts as a result. The bugs stay there because IBM provides us with no means of diagnosing problems in this subsystem. When there is a failure in shutting down or starting up, CP is too sick to take a dump of itself. IBM doesn't provide its VM customers with a standalone dump facility, so it gets very few dumps of shutdown and startup failures. If you begin seeing failures of this subsystem, I urge you to get hold of a standalone dump program from one of the other SCPs and take dumps and report the problems to IBM.

VII. Maintaining Multiple Systems

A. A Test System

A situation you are likely to run into sooner or later is one in which you have a fix from IBM assembled on your A-disk and you are not yet through testing it, but you suddenly have to build a new production system immediately because of some hardware change they forgot to tell you about. Or, it might instead be a local mod that you are not yet ready to install. Either way, getting your A-disk back to where it was before you started playing around with that new piece of code, so you can go ahead and build a new production system, is a process which can very easily go awry. This is why many people keep "accepted" local mods and locally-applied service on a disk accessed after the A-disk and keep only the volatile stuff on the A-disk. Another tip here is that whenever you are about to do an assembly which will end up replacing a textfile on your A-disk, it is a very good practice to first rename the old textfile:

```
 rename dmkiot txtptfs a = oldptfs a
```

It doesn't take up much disk space, and you may want it tomorrow.

A more general solution is to maintain two systems, a production system and a test system. To do this, you use the same minidisk layout we have been talking about, but you add a new control file:

FILE: DMKSPTST CNTRL

```
 TEXT MACS DKTSTMAC DKLCLMAC DKPTFMAC DMKSP DMKMAC DMSSP CMSLIB OSMACRO
 TST  AUXTST
 LCL  AUXLCL
 PTFS AUXPTFS
 TEXT AUXSP12
 TEXT AUXSP11
 TEXT AUXSP
```

The only differences here are that we have added DKTSTMAC at the beginning of the maclib list and we have added the statement TST AUXTST as the highest-level update specification. Then, when you get a questionable "fixtest" from IBM, you list it, not in an AUXPTFS auxfile, but in an AUXTST auxfile:

FILE: DMKDSP AUXTST A1

```
 *$*$*$*$*$*$*$*$*$*$*$*$*$*$*$*$*$ AUXTST $*$*$*$*$*$*$*$*$*$*$*$*$*$*$*$*$
 S13025DK - 11/04/81 - FIXTEST FOR TAPE I/O COMPLETING AFTER LOGOFF
```

You treat questionable new mods the same way. You specify DMKSPTST as the control file when you assemble the module, so that VMFASM will pick up the update from the AUXTST auxfile and build you a TXTTST textfile. When you invoke VMFLOAD, you specify DMKSPTST as the control file, and you end up with a CP nucleus that incorporates any TXTTST textfiles you may have sitting around. And you are in a position of being able to build a production system on a moment's notice by using your DMKSPLCL control file, so that you don't pick up any of the test versions of textfiles.

There is another use for the test control file and TXTTST textfiles. You may want to have permanently different versions of some modules in your test system, particularly if you test by running CP under CP. You may want to use a different version of DMKSYS in your test system, for example. In this case, you should build yourself an update which modifies your regular DMKSYS:

FILE: DMKSYS VIRTCP0 A1

```
 ./ R 5000 9000 $ 5100 100                                      VIRTCP0
          SYSOWN VMR901,VMM911,VMM913,VMM916,VMM917             VIRTCP0
 ./ R 11000 12000 $ 11100 100                                   VIRTCP0
          SYSRES  SYSVOL=VMR901,SYSRES=844,SYSTYPE=3350,        VIRTCP0*
                SYSCKP=2,SYSWRM=3,SYSNUC=4,SYSERR=6,            VIRTCP0*
 ./ R 17000 $ 17100 100                                         VIRTCP0
          SYSCOR  RMSIZE=4M                                     VIRTCP0
```

You should list this update in a DMKSYS AUXTST auxfile:

FILE: DMKSYS AUXTST A1

```
 *$*$*$*$*$*$*$*$*$*$*$*$*$*$*$*$*$ AUXTST $*$*$*$*$*$*$*$*$*$*$*$*$*$*$*$*$
 VIRTCP0 - 05/30/79 - DEFINE TEST SYSTEM 'SYSOWN', 'SYSCOR', AND 'SYSRES'
```

And you should use VMFASM with the DMKSPTST control file to create DMKSYS TXTTST.

```
 vmfasm dmksys dmksptst
```

Note that if your production DMKSYS textfile is named DMKSYS TEXT, then VMFASM just erased it for you.

B. The FRE013 Trap

One piece of code that you should definitely have in your test system, even if you do not have it in your production system, is the "FRE013 trap". The FRE013 trap (or the "FRE018 trap", as the AP/MP version is known) is a mod to DMKFRE which validates allocation and de-allocation of blocks of CP free storage. The trap abends CP with a FRE013 abend if a CP module tries to return a block that is larger or smaller than the block that was originally gotten or if a CP module tries to return a block that has already been returned. It also abends if the two-doubleword marker that it put at the end of the block has been overlaid; this is a good way to catch a function that is overwriting memory. In the majority of the cases in which the trap takes your system down, your system would have crashed very obscurely a few seconds later anyhow, so having the trap there just gets you a dump nearer the time of the error. The trap also marks each piece of free storage that is given out with its length and the address from which it was requested. When a block is returned, this marker is altered to reflect that fact. Without this trap or a similar one on your system, it is almost impossible to shoot "core cancer" bugs.

You get the FRE013 trap by phoning the Support Center and asking them to queue a request for the "free trap" to Level Two. You should seriously consider running this trap in your production system. My impression is that most of the old users use it all the time. The arguments against using it in your production system are that it uses some extra cycles (less than one per cent in my system), that it uses some real memory (about thirty pages in my 150-user system), and that, if you have a machine with the ECPS microcode, it turns off a portion of that microcode. It makes debugging CP a whole lot easier, though, so you are likely to have a more stable system if you use it. If the trap's memory utilization is a problem for you, you can modify it to add only one doubleword to each block, rather than two. If its CPU utilization is a problem for you, you can modify it not to set each new block to X'EE's. These two mods and a current version (February, 1982) of the FRE013 trap for SP are included in an appendix in the handout.

A note of warning: If you install the FRE013 trap on a system which has ECPS, be sure to make the additional changes documented in the trap for ECPS systems. If you do not, the results are predictable. I speak from experience.

If you decide to run the FRE013 trap only in your test system, the way to do it is to key in the trap and name it DMKFRE FRE013 (or some such), list that filetype in a DMKFRE AUXTST auxfile, and build a DMKFRE TXTTST textfile, using VMFASM with the DMKSPTST control file. Then, of course, you use that control file with VMFLOAD when you build your test system.

C. Systems for Multiple CPUs

Things are going along pretty smoothly for you now, until one day your boss comes in and tells you that they are so pleased with your work that they are going to give you another VM system to maintain. Don't despair. In VM, it is not much harder to maintain two systems than one, and you can use one set of service disks to maintain the systems for any number of CPUs. The CPUs can be a mix of UPs and dyadics and whatever. They can all have different I/O configurations and different logos and anything else you want to be different. Maintaining systems for two CPUs is just like maintaining one system for production and one system for testing. All it takes is a different control file for each CPU. With a unique control file for each CPU, you will be able to tell VMFASM to produce unique textfiles for each CPU, where that is required, as, for example, DMKRIO. And you will be able to tell VMFLOAD to build a unique nucleus for each CPU, incorporating the appropriate textfiles. I will illustrate with my own configuration. I am currently maintaining three systems (a 3033, a 4341, and a 4331) with one set of service disks. To do this, I use three control files:

FILE: DMKSPPU CNTRL A1

```
 TEXT MACS PUMAC PTFMAC DMKSP DMKMAC DMSSP CMSLIB OSMACRO
 3033 AUX3033
 PU   AUXPU
 PTFS AUXPTFS
 TEXT AUXSP11
 TEXT AUXSP
```

FILE: DMKSP41 CNTRL A1

```
 TEXT MACS PUMAC PTFMAC DMKSP DMKMAC DMSSP CMSLIB OSMACRO
 4341 AUX4341
 DIST AUXDIST
 PU   AUXPU
 PTFS AUXPTFS
 TEXT AUXSP11
 TEXT AUXSP
```

FILE: DMKSP31 CNTRL A1

```
 TEXT MACS PUMAC PTFMAC DMKSP DMKMAC DMSSP CMSLIB OSMACRO
 4331 AUX4331
 DIST AUXDIST
 PU   AUXPU
 PTFS AUXPTFS
 TEXT AUXSP11
 TEXT AUXSP
```

You will note that these three control files differ mainly in having a different highest-level update specification. The different control files are used to assemble and load CPU-dependent modules for our three machines, DMKRIO, DMKSYS, DMKSNT, DMKFCB, DMKBOX (the module which defines the VM logo), and our local charging algorithm module, DMKPXJ. So, I have, for example, DMKSNT TXT3033, DMKSNT TXT4341, and DMKSNT TXT4331 on my MAINT A-

disk. The other difference between the control files for the three machines is that the control files for the two distributed systems contain an additional update specification, "DIST AUXDIST", which is used to specify a number of local mods that we have found necessary to install only on our distributed systems. These mods are mostly fixes to make it easier to run an unattended system -- little things like not letting DMKDID fill up all of memory with messages about intervention being required on a printer.

You may find yourself getting carried away with making your systems different from one another, because it is so easy to do. You'll end up confusing both yourself and your users, though, so try to keep it under control. But, if you have multiple CPUs, you must be prepared to have some differences between them. For example, I have found myself in the situation of badly needing a certain PTF on the 4331 and not being able to install that PTF on the 3033 because it broke a function that was being used only on the 3033. In that case, I commented the PTF out of the AUXSP11 auxfiles and listed it in AUX4331 auxfiles, until I could get the problem it caused on the 3033 straightened out.

Many of us have concluded that the cleanest way to build different sysgen modules for different CPUs is to build a skeleton ASSEMBLE file, like this DMKRIO ASSEMBLE file:

FILE: DMKRIO ASSEMBLE A1

```
 RIO      TITLE 'DMKRIO - PRINCETON UNIVERSITY: VM/SP RELEASE 1'
          COPY  OPTIONS
 DMKRIO   CSECT
          END   DMKRIO
```

and to build a large update to this file for each of our CPUs, listing these updates in the CPU-specific auxfiles, of course. One of the advantages to this approach is that you don't end up with a DMKRIO TEXT for VMFASM to erase behind your back, but, rather, in my case, DMKRIO TXT3033, DMKRIO TXT4341, and DMKRIO TXT4331.

Another tip given to me by a friend who maintains systems for many CPUs is that he finds it useful to rearrange his loadlist so that the CPU-dependent resident modules are located at the end of the resident portion of the loadlist, just before DMKCPE. This has the advantage of making most of the module addresses the same for all of his systems (modulo any differences in V=R size, of course).

D. Maintaining Distributed Systems

Since I mentioned unattended, distributed systems above, this might be a good place to go into what I've learned about maintaining such systems. When your boss tells you he's giving you a distributed system, be sure to let him know he's also got to give you a terminal which will access that system. Even if the system is only a block away, you can waste an awful lot of time running back and forth through the rain otherwise. (It's always raining when you have to run back and forth.) We use PASSTHRU to

access one of our distributed systems and a locally-attached 3270 on a coax through a trench to access the other. Response is certainly better with the locally-attached tube, but either mechanism seems to be satisfactory, except for the fact that in neither case can I access the remote system from my home terminal (a TTY). Your management should realize that if they care at all about uptime on any of the systems for which you are the system janitor, then they must provide you with access to those systems from your home.

```
+--------------------------------------+
|          RULE NUMBER ELEVEN          |
+--------------------------------------+
|  YOU ARE ENTITLED TO A HOME TERMINAL |
+--------------------------------------+
```

Your management should also understand that proper maintenance of distributed systems requires powerful enough communications facilities to allow you to ship new CP and CMS systems down the line to the remote systems and to transfer dumps, monitor data, and accounting data from the remote locations to your central site. I have one friend who has had such severe problems with the reliability of communications with his remote systems that he has been forced to keep complete sets of CP textfiles at the remote locations so that new nuclei can be built there, but I don't think that he would recommend that mode of operation if it can be avoided. It's easier to keep all your service disks at the central site. If you ordinarily install your CP systems by IPLing a tape and if your remote sites have real operators and real tape drives, then you may find it satisfactory to distribute new systems by mailing tapes around. Otherwise, you'll probably find downloading new systems to be much better.

You'll almost certainly want to have the capability to read dumps remotely (most likely via PASSTHRU and IPCS), but you may still find it convenient to be able to transfer dumps to the central site for archiving and for mailing to IBM, or just to get better dump viewing performance.

One annoying little problem with installing new CP systems on an unattended, distributed processor is that IBM provides you with no way to IPL the new nucleus once you get it installed on the remote sysres. Either you must wait for the system to crash, in which case it will automatically re-IPL itself, or you must arrange for some human being to go to the remote site to do a SHUTDOWN and re-IPL. A number of installations have modified SHUTDOWN to add a REIPL option, which allows them to shut a remote system down and have it bring itself back up just as it would after a crash. Some of the rest of us just crash our remote systems on purpose to get the new nucleus IPLed. One easy way to do this is:

```
 stcp s7f 01
```

That puts an odd address into the I/O NEW PSW. It seldom lasts for long after that. There are some among us, however, who feel that it is more elegant to crash ones system by storing into the SVC NEW PSW.

VIII. The Differences Between CMS Service and CP Service

At long last, we come to CMS. CMS service is done the same way CP service is done, but, because CMS has a more complex structure than CP has, there are a few additional tasks for you to master. Specifically, you must learn to build the CMS system disks (the "S-disk" and the "Y-disk"); you must learn to build the CMS saved systems; and you must learn to build the CMS MODULE files.

A. CMS Macro Update and Auxfile Names

There is also one minor difference from CP which I will tell you about now before I forget about it. The auxfile and update names for fixes to CMS macros and copy sections do not follow the standard rules. For VM/SP Release 1, the CMS macro updates have filetypes of EnnnnnDS, rather than SnnnnnDS, as you would expect, and the macro auxfiles are named AUXMSP, rather than AUXSP. Therefore, you must use a different control file for updating CMS macros and copy sections than you use for updating CMS ASSEMBLE files. The one IBM supplies for updating the SP1 CMS macros is:

FILE: DMSMSP CNTRL S2

```
 TEXT MACS
 TEXT AUXMSP12
 TEXT AUXMSP11
 TEXT AUXMSP
```

The reason it is done this way is that two CMS macros have the same filenames as CMS ASSEMBLE files, and that wouldn't work if the updates and auxfiles had the same filetypes for both macros and source. This is a totally unnecessary complication, which the developers really should do something about. It can be a real gotcha, because when you want to find out what parts of CMS were hit by APAR VM12345, it's so easy to do a LISTFILE on * S12345DS and to forget also to do a LISTFILE on * E12345DS.

B. Another Maxim

This also seems like a good time for another maxim:

```
+-----------------------------------+
|        RULE NUMBER TWELVE         |
+-----------------------------------+
|  CHANGE ONLY ONE THING AT A TIME  |
+-----------------------------------+
```

Most new users seem to have been told that the right way to do VM service is to apply service to all the components at once. Slap up the PUT and let VMSERV go wild. The old users, however, make a point of not doing that. We NEVER install new versions of CP and CMS at the same time, because if you do them one at at time, it is easier to track down the bugs, it is easier to back out, and it is just plain easier. Furthermore, many of us

prefer to make both old-release and new-release CMS systems available simultaneously during a transition period.

For years, IBM has been telling us that we MUST install new releases of CP and CMS simultaneously, and for years we have been refusing to do so. So, it's been left to the user community to prepare and distribute the fixes necessary to run a new CMS under an old CP and vice versa. You'll find that VMSHARE will provide you with a lot of guidance in this area. Even though it may involve your applying mods to your system, I still advise you not to install new releases of CP and CMS simultaneously on a system of any complexity. You may, of course, need to install a new CP nucleus with a new DMKSNT when you install a new CMS, but you can plan ahead so that there is nothing else new in that CP nucleus. It's a shame to have to back out of a new CMS because of some unrelated glitch in CP. A corollary to this rule suggests that you will be better off keeping your service disks for various components separate, i.e., don't put CP and CMS fixes on the same disk. Incidentally, it appears that SP1 and SP2 CMS and CP mix much more easily than has been the case with previous new releases; IBM is to be congratulated for this.

C. CMS Structure

CP has an extremely simple structure. With only a few exceptions*, all CP modules are part of the CP nucleus, and the CP nucleus has the simplest possible structure -- a straight line -- as described by the CP loadlists. CMS has a more complex structure than CP. Many CMS modules are included in the CMS nucleus, which is described by the CMS loadlist, CMSLOAD EXEC. When you IPL the CMS nucleus, however, you do not bring all of CMS into memory, because not all of CMS is in the CMS nucleus. The portions which are not in the nucleus are in the form of MODULE files and textfiles, which are disk-resident. The disk that they reside on is the CMS system disk, the S-disk, which is normally at virtual address 190 for a CMS user. When you are using CMS and you invoke a CMS function which is not part of the CMS nucleus, the appropriate module is loaded into memory from the S-disk and is then executed.

However, if you had a lot of users bringing copies of the same CMS nucleus and the same CMS disk-resident modules into their own virtual memories, you would need a lot of real memory to hold all that virtual memory. Since much of the code in CMS is re-entrant, there is no reason why one copy of it could not be shared by many users. That is what the CMS saved systems are for. They allow named copies of the re-entrant portions of CMS to be stored on CP-owned volumes so that they can be loaded into memory to be used by many users at once. Furthermore, they are loaded by efficient CP paging I/O, so there is an advantage to using saved systems even for non-

---

* A few CP modules (DMKDDR, DMKDIR, DMKFMT, DMKLD00E, and DMKRND) are not part of the CP nucleus. Instead, they are used in the form of "standalone" utilities which you can IPL in either a virtual machine or a bare machine. Some of them (with the notable and annoying exception of DMKFMT) are also available in the form of MODULE files which execute under CMS.

sharable code. Thus, your CMS user doesn't ordinarily IPL 190 to bring in the CMS nucleus; instead, he IPLs a saved CMS nucleus, which you have built and named "CMS". This saved CMS nucleus contains two segments, 0 and 1. Segment 0 is not shared between users, but segment 1 is. You may also have built two other saved systems, the CMS "shared segments", CMSSEG and CMSZER. CMSSEG contains much of the S-disk-resident CMS code, such as the editors and OS simulation. CMSZER contains sharable code which is moved from segment zero of the nucleus, plus the file directories for the S-disk and the Y-disk, which are known as the "SSTAT" and the "YSTAT". The CMS nucleus knows how to load these shared segments and will use them, rather than the S-disk-resident code, as long as the segments exist and are defined as having virtual memory locations beyond the user's virtual machine size. You don't have to have saved systems to run CMS, but if you intend to have more than one CMS user at a time, it's a good idea, so I will be assuming the use of the CMS saved systems.

There are usually a good many functions available to CMS users which are not, strictly speaking, a part of CMS, as, for example, the various language processors. These are ordinarily resident on the CMS Y-disk, which is defined at address 19E for each of your CMS users. IBM and other software vendors are slowly coming around to designing compilers and other big packages to run in "shared segments" similar to CMSSEG and CMSZER. So, it may be that much of your Y-disk-resident code can also be executed in the form of shared segments. This is an area where you will find lots of user mods, too, because the performance benefits are quite noticeable.

One of the things that make the S-disk and the Y-disk different from regular CMS user-type disks is that only the mode 2 files are visible when those disks are accessed as S and Y. This is achieved by building the S-disk and Y-disk file directories with the equivalent of the commands ACCESS 190 S/S * * S2 and ACCESS 19E Y/S * * Y2. This is done for performance reasons; smaller file directories can be scanned more quickly and require less memory. There are lots of mode 1 files on the S- and Y-disks, but the general CMS user doesn't need to see them. The mode 1 files are such things as textfiles which the user doesn't reference directly because he instead uses the MODULE files into which those textfiles have been "link-edited". In other words, the mode 1 files on the S- and Y-disks are on those disks because they are needed to build CMS and the compilers and so forth, but they are stored invisibly, as mode 1 files, because nobody should ever want to use them. But you need those mode 1 files for doing CMS maintenance, so you will ordinarily have the S-disk accessed at another mode, so that you can "see" the mode 1 files. You will occasionally want a similar access to the Y-disk.

Another unusual property of the S- and Y-disks is that they contain "auxiliary directories". Auxiliary directories are another way of keeping the users' S- and Y-disk directories small. An auxiliary directory is contained in a mode 2 file, such as ASSEMBLE MODULE S2. It points to the disk location of a bunch of mode 1 files needed only by that particular function, as the assembler MODULEs, in this case. When the user invokes a function which uses an auxiliary directory, the auxiliary directory is appended to the main directory in memory, but only for the duration of the command. Ordinarily, the only auxiliary directory on the S-disk is the one for the assembler. There may be several on the Y-disk, depending on which

language processors you have. The existence of the auxiliary directories makes copying your S- and Y-disks a bit tricky. If you use COPYFILE to copy your S-disk to a new location, for example, you will need to rebuild the assembler's auxiliary directory, because the relative addresses of those mode 1 files that the assembler auxiliary directory knows about will tend to be different on the new disk. The same would hold for copying your Y-disk with COPYFILE, of course. The GENDIRT command is used to build an auxiliary directory; it's not hard to do, but it is hard to remember to do. If you copy your entire S- or Y-disk with DDR COPY ALL, the auxiliary directories do not have to be rebuilt, because all of the files on the new disk will have the same relative address that they had on the old disk. (However, it's a good thing to compress your system disks now and then, to reduce head movement, so it's worth doing the GENDIRTs now and then.)

You need to understand the structure of the S-disk, so that you can build yourself a new one from scratch when you need to. For the most part, the S-disk is just another CMS-format disk, but it has to be able to be IPLed, so it has IPL text at the beginning and the CMS nucleus at the end. When you build a CMS nucleus, you normally specify that there should be IPL text written on cylinder/block zero of the S-disk to point to the cylinder/block at which the nucleus begins. When your users IPL 190, this cylinder-zero IPL text is loaded, and then it loads the actual CMS nucleus. When you get yourself a new S-disk, you CMS FORMAT it and then you use the RECOMP option of FORMAT to decrease the apparent size of the disk. The space left at the end is where the CMS nucleus is kept. RECOMPing the size to below the nucleus prevents regular files from overlaying the nucleus. You should look in the sysgen manual and the PUT Memo to Users to find out how big your S-disk should be and how much room you should leave at the end for the nucleus. You should also be aware of a bizarre restriction for FBA S-disks: the nucleus must begin on an FBA block whose block number is an even multiple of 256 (counting from the beginning of the S-disk, not from the beginning of the physical volume).

I recently learned a neat trick from one of the CMS old-timers. He maintains a rather volatile CMS system and has found it useful to have alternate CMS nuclei, similar to the alternate CP nuclei we talked about earlier. Of course, not all of CMS is contained in the CMS nucleus, so you don't back all the way out of a bad CMS by IPLing an alternate CMS nucleus, but, if you've done something really disastrous to CMS, it's likely to be something in the nucleus that's wrong, so that alternate nucleus may be quite valuable. What my friend does is quite simple; putting an alternate nucleus on his S-disk requires no mods at all. He allocates his S-disk so that it is big enough to hold the extra nucleus. He RECOMPs enough room at the end for two nuclei, rather than one. He builds his public, production nucleus in the standard way, in one of the two nucleus slots, specifying that the cylinder-zero IPL text should point to it, so that anyone who simply IPLs 190 gets that CMS nucleus. He builds his reliable, backup nucleus in the other slot, but replies that he doesn't want the cylinder-zero IPL text rewritten, thus leaving it pointing to the other nucleus. But, if it turns out that he needs to IPL the backup nucleus, he can issue an IPL command which specifies the cylinder/block which is to be IPLed (the one where the backup nucleus starts), as, for example, "IPL 190 43".

D. Recommended Control Files and Service Minidisks for CMS

I will be using these control files for CMS:

FILE: DMSSPLCL CNTRL A1

```
 TEXT MACS DSLCLMAC DSPTFMAC DMSSP CMSLIB OSMACRO DOSMACRO TSOMAC
 TEXT AUXLCL
 TEXT AUXPTFS
 TEXT AUXSP12
 TEXT AUXSP11
 TEXT AUXSP
```

FILE: LCLMAC CNTRL A1

```
 TEXT MACS
 TEXT AUXMLCL
 TEXT AUXMPTFS
 TEXT AUXMSP12
 TEXT AUXMSP11
 TEXT AUXMSP
```

As you can see, they differ from the IBM-supplied control files only in specifying auxfiles and maclibs to contain locally-applied service and local mods.

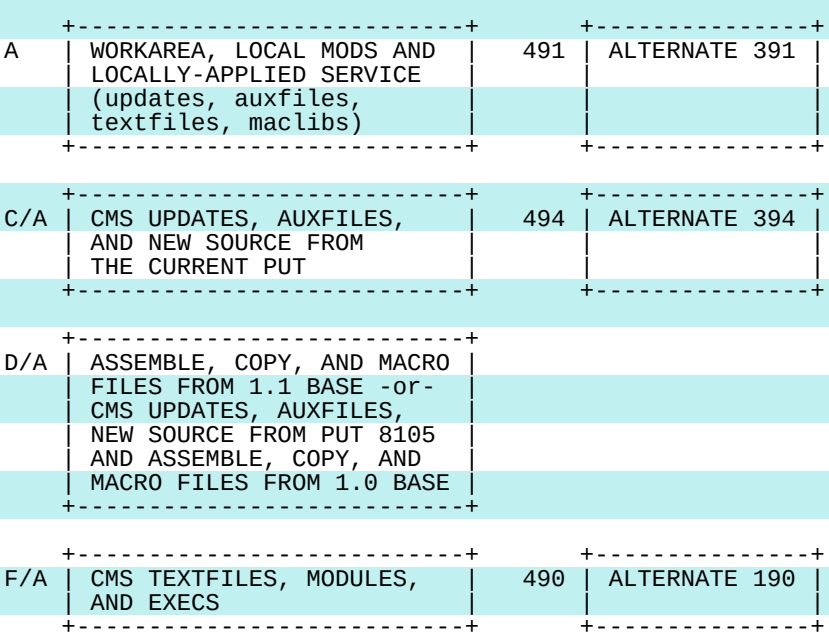
Since the CMS system disk, the S-disk, traditionally contains all CMS textfiles and MODULEs, even the old users generally give in and overlay local mods on top of service on top of base on the S-disk. They compensate for the uncleanness of this by keeping alternate S-disks to flip-flop from PUT to PUT. (If they modify CMS much, they tend also to keep around a totally unmodified S-disk, at the current PUT level, for deciding whether it's their bug or IBM's.) IBM's VMSERV EXEC is designed to allow you to install CMS service without having an alternate S-disk. If you cannot afford a second S-disk, then I suggest that you use VMSERV to install CMS service during dedicated test time. I also suggest that you pray a lot while you are doing it. You will be running CMS from your S-disk while you are writing the new service onto that S-disk. If you get a machine check or a power failure or a bug in VMSERV while you are doing this, it is likely to leave you without a usable S-disk, which is the same as being without a system at all. This brings up my next-to-the-last rule:

```
+--------------------------------------+
|         RULE NUMBER THIRTEEN         |
+--------------------------------------+
|  YOU CAN NEVER HAVE TOO MANY S-DISKS  |
+--------------------------------------+
```

If you have only one S-disk, you are going to have to have a lot of dedicated test time for installing CMS service and new releases. If you have two S-disks, you can put a new version of CMS into production by just taking CP down and bringing it back up with a different DMKSNT. You are made much less vulnerable to having an installation problem cause a long outage, and you can have new versions of CMS available for your users to

try out in advance of their being put into production. I hope you can convince your management that the sensible approach is to let you have two S-disks. If they won't let you have more than one, be absolutely certain that you have a couple of good DDR dumps of your S-disk and a working version of an IPLable DDR, either on tape or on cards.

For the rest of this discussion, I am going to assume that you have two S-disks, the production version at address 190 in MAINT's virtual machine, and the alternate one at address 490. At any one time, 490 may be either an old CMS system you may still want to back out to or a new CMS system that you are not yet ready to put into production. As we did in the case of CP, let's start with the assumption that you are a brand new user who has somehow got SP CMS up and running. This will be the assumed service minidisk layout, but what I'll be doing would also work with the IBM minidisk layout as long as you have an alternate A-disk and an alternate S-disk:



Note that the A-disk for CMS service in this example is at address 391. The service disks for the current PUT and for the base are normally accessed as read-only extensions, as is the S-disk. The virtual machine you use for doing CMS maintenance will ordinarily have a write link to 190 defined in its directory entry, but your PROFILE EXEC should re-LINK 190 read-only, both to reduce the chances of your inadvertently modifying the S-disk and also to prevent your users from receiving a message about the LINK every time they log on when you're logged on.

Before we go any further, I should point out that there are hard-line purists among us who insist that most of the mode 1 files on the S-disk and the Y-disk should not be there. The quality of the performance your CMS users get depends a lot on how quickly they can load files from the system disks. Obviously, the larger these disks are, the greater is the average head movement required to service a user's requests for system files. There are some sophisticated installations which do run with S- and Y-disks from which most of the mode 1 files have been removed to the base, service, and local mods disks, as with CP. There are people here who run with 5-cylinder S-disks. Implementing such a departure from the standard configuration is beyond the scope of this presentation, but you should be aware of the possibility.

You should certainly keep in mind the desirability of keeping your S- and Y-disks as small as possible and as unfragmented as possible. And, as a general rule, you should be careful not to "pollute" your S-disk with anything that is not an integral part of CMS. One good move IBM made in VM/SP was to suggest that the EREP libraries be removed from the S-disk to a separate disk at virtual address 201. Since the EREP libraries account for about a fourth of the space required for the S-disk and tend to be used once a day by one userid, this recommendation made a lot of sense. You may want to consider removing other functions too, e.g., the vanilla IPCS, if you're using the program product, or all of the DOS stuff, if you can get away with it. Also, if your S-disk was built originally with either DISK LOAD or VMFPLC2 LOAD, you will find that copying it to a TDISK and back again with COPYFILE may reduce the space used by several hundred blocks. This is because of lamentable bugs in both VMFPLC2 and DISK which leave empty blocks at the ends of files.

IX. Installing Corrective Service for CMS

A. Regenerating a CMS Module

You are sitting in your office one day, browsing through a new PUT bucket that just came in the mail, when a guy from down the hall sticks his head in and tells you that SSERV is broken; it keeps getting an opcode exception. If you are like me, you tell him that that's too bad but that you have never heard of SSERV. Then he tells you that it's part of CMS, a really neat DOS utility. You tell him you'll get right on the problem. After you look up SSERV to find out what it does, you decide to take a look at that new bucket before calling the Support Center. And there, in the General Information section of the SPCMS subset, you find an exact description of the problem:

> "THE SSERV MODULE AND THE DMSSRV TEXT ARE INCORRECT ON PUT
> 8107. THERE IS AN MVC INSTRUCTION MISSING WHICH WILL CAUSE
> A PROGRAM CHECK OPERATION EXCEPTION.
> RECOMMENDATION: REASSEMBLE DMSSRV AND CREATE A NEW TEXT.
> THEN PERFORM A CMSGEND SSERV TO CREATE A
> NEW SSERV MODULE."

Sounds like they left some debugging code in, doesn't it? This is a typical packaging problem, and the fixup is typical, too. You don't even have to apply a PTF; you just have to pick up the pieces and put them back together. The instructions tell you to reassemble DMSSRV. That is certainly easy enough:

```
vmfasm dmssrv dmssplcl
UPDATING 'DMSSRV ASSEMBLE D1'
APPLYING 'DMSSRV S12530DS D1'
ASMBLING DMSSRV
ASSEMBLER (XF) DONE
NO STATEMENTS FLAGGED IN THIS ASSEMBLY
DMSSRV TEXT CREATED
```

That gives you a file called DMSSRV TEXT on your CMS service A-disk, 391. But what do you do then? Well, the bucket writer was nice enough to tell you. He said to create a new SSERV MODULE file by doing a CMSGEND SSERV. You type "CMSGEND SSERV", and you get a response indicating that CMSGEND is not a known CP/CMS command. But, I happen to know where CMSGEND is, because I have gotten that message so many times. CMSGEND is an EXEC, and it lives on the S-disk. The reason you couldn't find it is that it is a mode 1 file and you forgot about accessing the S-disk at another mode, so you can't see any of the mode 1 files. IBM makes CMSGEND a mode 1 file for a good reason -- CMSGEND ordinarily needs to pick up mode 1 textfiles from the S-disk in order to function properly; keeping CMSGEND at mode 1 forces you to have the S-disk accessed properly when you use it. You don't need write access; you just need to access 190 as something other than S and try again:

```
access 190 f/a

cmsgend sserv

*** CURRENT STATUS:
FILE ' SSERV MODULE A2 ' DOES NOT EXIST
FILE ' SSERV MODOLD A1 ' DOES NOT EXIST
*** LOADING:
 INVALID CARD - S12530DS 104 UV03791 MISSING STATEMENT IN SOURCE BOOK.
 INVALID CARD - *         DMSSRV   S12530DS D1 CMS495  02/17/81   16:10:00
 INVALID CARD - *         DMSSP    MACLIB   S2 CMS190   6/10/81   18:45:00
 INVALID CARD - *         CMSLIB   MACLIB   S2 CMS190   2/07/81   12:19:00
 INVALID CARD - *         OSMACRO  MACLIB   S2 CMS190   3/18/81   13:32:56
 INVALID CARD - *         DOSMACRO MACLIB   S2 CMS190   6/11/80   13:25:00
 INVALID CARD - *         DMSSRV   ASSEMBLE A1 CMS496   6/13/80    7:51:00
 DMSSRV   SD 020000

*** RESULTS:
' SSERV MODULE A2 ' CREATED FROM TEXT DECK ( S ) DMSSRV
WITH ATTRIBUTES CLEAR NOMAP ALL
R;…
```

You invoke CMSGEND, which invokes LOAD, which puts out messages complaining about all those comment cards in your textfile, but the MODULE gets built anyway. (You could suppress the messages by specifying the NOINV option of CMSGEND, but I prefer the security of seeing the messages.) Now you have on your A-disk a new SSERV MODULE which incorporates that new DMSSRV TEXT from your A-disk. You DISK DUMP the MODULE to the guy down the hall and ask him to test it. He does. It works fine. That's great, but the only places where the good version exists are on your A-disk and his A-disk, so you are going to have to do something to make it generally available, i.e., you must put it on the S-disk where everyone can find it.

But before we get into that, let's talk some more about CMSGEND. CMSGEND is a very nice, reliable tool that understands how to build the MODULE files for various CMS commands from their constituent textfiles. It is also used to build the MODULEs that are built from CP textfiles, e.g., DDR, DIRECT, etc. When you use CMSGEND, you just tell it the module name. It knows which textfiles are needed and what GENMOD options to use in each case and whether or not the MODULE should be generated to run in the CMS transient area. The CMSGEND EXEC is sort of interesting to look at, but you needn't bother to, because you are quite safe in treating CMSGEND as a primitive. The only problems I have ever had with CMSGEND were the result of its being so badly documented in the sysgen manual.

Some of what the sysgen manual tells you about CMSGEND is both wrong and dangerous. It tells you that to use CMSGEND you must access your S-disk as your read/write A-disk. This is certainly not true and not something you ever want to do when there is some good way to avoid it. The manual goes on to tell you that if you want to generate GLORP MODULE from a DMSGLP textfile that is on some disk other than your S-disk, then you should access your 190 as A, access the other disk as B/A, temporarily rename the DMSGLP TEXT that is on the 190 to something else (so CMSGEND will not find it and will use your new version instead), and then invoke CMSGEND. CMSGEND will then erase GLORP MODOLD, rename GLORP MODULE to GLORP MODOLD, and create a new GLORP MODULE, all on the 190. Don't believe it! That is, indeed, what will happen if you take those steps, but that is NOT the way to do it. As we have seen, CMSGEND works perfectly well without your having write access to the S-disk, and it is perfectly legitimate to want to build CMS MODULEs from textfiles that are not on an S-disk. Indeed, that is the best way to test fixes to CMS MODULEs, as well as fixes to DIRECT and DDR.

There is, however, one important point about using CMSGEND that did not come out in our very simple example of regenerating SSERV. CMSGEND doesn't use a control file. This means that it doesn't know about textfiles that have filetypes of TXTLCL or TXTPTFS or TXTTST or whatever. It just knows about textfiles that are named TEXT. The DMSSRV textfile we built above was named TEXT, so it all worked out, but it would not have worked if that had been a DMSSRV TXTPTFS textfile. This inability to use control files is unfortunately not a problem just with CMSGEND; it occurs throughout CMS. The textfiles which are incorporated into MODULEs must be named TEXT because CMSGEND doesn't understand control files; the textfiles which are incorporated into the shared segments must be named TEXT because CMSXGEN and CMSZGEN don't understand control files; and the textfiles which sit out in the open as mode 2 files on the S-disk must be named TEXT because CMS LOAD doesn't understand control files. Some CMS nucleus textfiles could have filetypes other than TEXT, since the CMS nucleus is built with good old VMFLOAD, but it is safer always to follow:

```
+-------------------------------------------+
|            RULE NUMBER FOURTEEN           |
+-------------------------------------------+
|  IF IT'S CMS, YOU'VE GOT TO NAME IT 'TEXT'  |
+-------------------------------------------+
```

This also holds for the CP textfiles for DMKDDR, DMKDIR, DMKFMT, etc., when you want to use them to build IPL decks or MODULEs.

This is why all the update level identifiers in my recommended CMS control files are TEXT. This way, any time you assemble a CMS module, VMFASM will name the output TEXT, rather than TXTPTFS or some such. So, you have to remember only thirteen rules, not fourteen. But, if you are going to be modifying CMS a lot, you might be better off to use a more sophisticated control file and keep the modified textfiles on your A-disk named TXTPTFS and TXTLCL and TXTTST and so forth. However, you will have to remember to rename them when you copy them to the S-disk or when you are building MODULEs on your A-disk.

B. Putting It on the S-disk (Or on the Y-disk)

Now, back to the question of getting the new DMSSRV TEXT and the new SSERV MODULE onto the S-disk. One thing you should understand about changing the S-disk, is that the S-disk directory is saved in both the CMS saved system and the CMSZER shared segment, so you must resave both of those when you change the S-disk. The Y-disk directory is saved only in CMSZER; therefore, you need not resave CMS when you change the Y-disk, but you must update the saved Y-disk directory in the CMSZER shared segment.

Here are the steps you need to take to put your new SSERV into production, starting with the new textfile and MODULE on your A-disk:

```
link * 190 190 mr
access 190 f

rename dmssrv text f1 = oldtext f5
copyfile dmssrv text a = = f1 ( olddate
copyfile sserv module a = newmod f1 ( olddate
rename sserv module f2 = oldmod f5
rename sserv newmod f1 = module f2

ipl 190 clear
VM/SP CMS - 02/28/82 16:36
savesys cms

VM/SP CMS - 02/28/82 16:36
Y (19E) R/O
R;

dmszes
IS THE SHARED S-DISK DIRECTORY TO BE USED WITH THIS SYSTEM? YES|NO
yes
IS THE SHARED Y-DISK DIRECTORY TO BE USED WITH THIS SYSTEM? YES|NO
yes
SYSTEM SAVED
R;
```

You first get write access to 190. You rename the old textfile on the 190, rather than erasing it, because you never know when you might want it. But you make it mode 5, so that the users can't see it and so you'll know that it's garbage which you can erase one of these days. Remember that you want to avoid fragmentation of the S- and Y-disks for performance reasons, so don't let the obsolete files accumulate endlessly. You copy over the new textfile, making it mode 1 (since that's what the old one was) and you copy over the new SSERV MODULE under a temporary name. Then you quickly rename the old SSERV MODULE S2 to SSERV OLDMOD S5 and rename the new one SSERV MODULE S2. You do an IPL 190, which (unlike IPL CMS) causes the nucleus SSTAT to be rebuilt. You then immediately do a SAVESYS to rebuild the CMS saved system, incorporating this new SSTAT. (You enter that SAVESYS command when CMS puts up its first read, not later, because that is the state in which you want CMS saved.) Once that is done, you use the DMSZES command to resave the SSTAT and YSTAT portions of CMSZER, in order to get the SSTATs (the S-disk directories) in CMSZER and CMS back into synch with

one another. If the new file had been going onto the Y-disk, the procedure would have been the same, except that you would have skipped the SAVESYS.

Be very careful, when you move something to the S- or Y-disks, to make sure you make it mode 1, if it is supposed to be mode 1, and that you make it mode 2, if it is supposed to be mode 2. If you make something mode 1 that is really supposed to be mode 2, the users will get strange error messages and start phoning you. The easiest way to tell which mode a file is supposed to be is to look at the file it is going to replace.

In this example, I used the DMSZES command to rebuild the SSTAT and YSTAT in CMSZER. Even though only the S-disk was changed, you must tell DMSZES that you want both the SSTAT and the YSTAT rebuilt. Contrary to what you might expect, if you tell DMSZES to rebuild only the SSTAT (because you've changed only the S-disk), it will wipe out the pointer to the CMSZER YSTAT when it rebuilds the SSTAT. (There is a recent fix for this problem, VM15270.) On the other hand, if you've changed only the Y-disk and you tell DMSZES not to rebuild the SSTAT, it exits without building either the SSTAT or the YSTAT. The CMSZGEN EXEC invokes DMSZES for you when you use it to build CMSZER. So, another approach would have been simply to rebuild the entire CMSZER shared segment, using the CMSZGEN EXEC, but DMSZES is a bit faster and you don't have to increase your virtual machine size to use it. Note that when you use DMSZES, your virtual machine size must be small enough that CMSZER can be attached beyond the end of your virtual machine, as it normally is for a CMS user; however, when you use CMSZGEN, your virtual machine size must be large enough that CMSZER can be loaded into your virtual machine.

You should be aware of the fact that the standard one-segment CMSZER has room for only seven hundred File Status Table entries. If you have more than a total of seven hundred mode 2 files on your S- and Y-disks, then you will overflow CMSZER, which will cause you to get an obscure error message when you invoke DMSZES or CMSZGEN:

```
 DMSZES100W CMSZER SYSTEM NAME 'CMSZER' NOT INITIALIZED
```

If this happens, you should question whether you really need all those files on your system disks. If you do, you can redefine CMSZER in your DMKSNT to be two segments long. This will give you room for an additional 1024 mode 2 files. (It may also mean that you have to change the location of your CMSSEG so that CMSSEG and CMSZER don't overlap in virtual memory.)

C. A Digression on the Subject of Updating a Production CMS System

The first several times you go through this operation, you certainly want to do it with no other users logged onto your system using the S-disk. In fact, you may want never to do it any other way. This is an area in which there are many schools of thought. Some installations take the position that it is wrong ever to change anything about the production CMS while there are users logged on, because the chances are so good that you will interfere with their use of the system. Others say you are hurting the users more by taking the machine away from them every time you need to put some little fix on CMS. A few of the larger, older installations have solved the problem by means of elaborate local mods which make it completely safe for them to update CMS "on the fly". (Perhaps the most notable of these mods are the ones that Charlie Brown did when he was at TYMSHARE. I wish he had done them at IBM instead.) Most of the rest of us take the position that we will change CMS on the fly when we have to, using the most reliable mechanisms that we can, and that when the change is major, we will install it by flip-flopping our S-disks and saved systems across a CP IPL. In my examples, I am going to show you how to update CMS on the fly, but I am not necessarily recommending that you should do it that way. The same command sequences are reasonable to use even if you are the only one on the machine.

The situation is this: the users who are logged on have access to 190 and 19E and have the S-disk and Y-disk directories in memory. They will continue to use those directories after you change the disks, so you must make sure that their directories remain valid; otherwise, the users will get strange I/O error messages. Their directories can "see" only the mode 2 files on the S- and Y-disks, so you can change any of the mode 1 files which are not pointed to by auxiliary directories (except for a problem with HELP, which I'll explain later). You must not erase or replace a mode 2 file, but you CAN change its name and mode, as long as you leave the file in the place where the old directory expects to find it. In our SSERV example, the only mode 2 file that is changed is the SSERV MODULE. You rename that to SSERV OLDMOD S5 and copy over a new SSERV MODULE S2. This allows the users who are already logged on to continue to use that old version of SSERV. It still exists at the old location, and their file directories still point to it.

You will recall that the S-disk directory (the SSTAT) is kept in two places. The saved CMS system contains an SSTAT in segment zero. When a user IPLs CMS, this SSTAT comes into his virtual machine with the rest of the CMS nucleus. CMS then tries to attach the CMSZER shared segment. If that is successful and if you have saved an SSTAT in CMSZER and if the two SSTATs match, then CMS releases the memory used by the SSTAT in nucleus segment zero and uses the shared SSTAT in CMSZER instead.

Getting back to our example, until you change the SSTAT in the CMS saved system by doing the SAVESYS, anyone who logs on will still be able to use the saved SSTAT in CMSZER, which points to the old SSERV MODULE. Once you do the SAVESYS, everyone who logs on or re-IPLs CMS will get the new S-disk directory in the new CMS saved system and, thus, the new SSERV. However, users who logon (or re-IPL) between the time that you do the SAVESYS and the time that you do the DMSZES to update the S-disk directory in the CMSZER shared segment will have two S-disk directories which will not match. CMS initialization will notice this and will not use the shared directory in CMSZER. It will also issue the error message:

```
DMSINS100W CMSZER SYSTEM NAME 'CMSZER' SSTAT NOT AVAILABLE.
```

(There is a similar message when you change the Y-disk and don't update CMSZER's YSTAT.) Users who never notice the most urgent LOGMSGs will notice this message and get upset. So, you should resave the shared directory as quickly as possible. One little trick that will speed the process up a bit is to stack the replies to DMSZES:

```
dmszes#yes#yes
```

The situation with changing the Y-disk is slightly different. The Y-disk directory is not saved in the saved CMS system, so every time a CMS user logs on or re-IPLs CMS, he gets a new nucleus copy of the Y-disk directory, which is built from the Y-disk as it exists at that moment. So, you might have a problem with someone who logs on between the time an old Y-disk MODULE gets renamed to mode 1 and the time the new mode 2 file is created. As far as this user can tell, there is no such file on the Y-disk, because the copy of the Y-disk directory in his nucleus doesn't list such a file, since it lists only mode 2 files. This is a hole that is simply there, and you cannot stop it up without modifying the system. You can, however, make the hole as small as possible by doing the RENAMEs very quickly in an EXEC, rather than keying in the commands by hand. This can still catch someone, of course, but the hole is rather small.

Even if you don't catch someone that way, you have the problem of DMSINS100W messages for the Y-disk too. When a user logs on or re-IPLs CMS after you have changed the Y-disk, but before you have rebuilt the shared YSTAT in CMSZER, he'll get that message telling him that he can't use the shared version of the YSTAT because it doesn't match the YSTAT that has just been built in his CMS nucleus. Because there is no <u>saved</u> nucleus version of the Y-disk directory, users will start getting the YSTAT NOT AVAILABLE message at logon as soon as you change the Y-disk. This will continue until you get the DMSZES done to get the shared YSTAT back in synch with the real volume. So, you would be well advised to have an EXEC for updating the Y-disk. Basically, your EXEC should copy the new file to the Y-disk under a temporary name, then rename the old file to OLDxxx Y5, rename the new file to its proper name and mode 2, release the disk to rewrite the directory, re-access it to build a new YSTAT in your nucleus (ACCESS 19E Y/S * * Y2), and then do the DMSZES, as above.

If you were to take these actions at a time when there weren't a lot of people logging on, it might be acceptable to have a few users get the SSTAT or YSTAT message and non-shared directories. There is another, more severe problem with changing the S-disk, however. The users who were logged on before all this started are still using the old version of the CMS saved system and the CMSZER SSTAT and YSTAT and will continue to do so until they re-IPL or log off. If they use the HELP facility, they may have trouble because of the way the HELP files are accessed. The HELP files are mode 1, not mode 2, so they don't show up in the saved directories, but the HELP function uses the saved S-disk directory in the CMS nucleus or in CMSZER to point to the disk-resident directory for the disk, so that it can find the mode 1 HELP files. If you've done anything to change the disk-resident directory, then the memory-resident directory may not point to a valid disk-resident directory. More precisely, the "disk origin pointer" in virtual storage points to either record 4 or record 5 on the disk. One of these two records always points to the current directory file; the other points to the previous directory file. Each time the disk directory is updated, the roles of records 4 and 5 are reversed. So, if your saved S-disk directory points to record 4 as the disk origin pointer, and you make any change to the S-disk, then record 5 will become the valid disk origin pointer, but your saved directory in virtual storage will still point to record 4, which will point to the old directory. There is no guarantee that the old directory will remain valid. In this case, users trying to get help will get I/O error messages instead. Some installations have addressed this problem by such means as having the HELP XEDIT profile (HELPXED XEDIT) automatically re-access the S-disk when an I/O error is received. (Note that you can't re-access 190 as S, so the fixup in this case would be to access it at some other mode.) You may have mode 1 HELP files on your Y-disk, as well as on your S-disk. Some program products put mode 1 HELP files there. If you have such files, then the situation with changing the Y-disk is the same as the situation with changing the S-disk.

There is one other potential problem with updating CMS with users logged on. Your logged-on users are using the CMS shared segments, CMSSEG and CMSZER. If you make a change to CMS which requires you to resave one of these, then the system will automatically give each of the logged-on users a private copy of the old shared segment, which he will continue to use until he logs off or re-IPLs CMS or detaches the segment. This doesn't help your performance, if you do it in the middle of the day, of course, but there is potentially a worse problem. Those old users will get copies of the old segment, but they'll get copies of only those pages which have been referenced by someone since the last CP IPL (or the last time the segment was saved). After you save the new shared segment, if one of these old users references a shared-segment page which had not been referenced previously, then he will get that page from the new shared segment. I'm sure you can see the possibility of getting bizarre results from this. This is less likely to be a problem with CMSSEG than with CMSZER, because CMSSEG is detached after every use (unless you have modified your system not to do that, as many installations have). In fact, in practice it is seldom a problem with either segment. If several people have been using CMS for a while before you resave the shared segment, it's very likely that all the pages which are ever going to be referenced will already have been referenced before you resave the shared segment. Nevertheless, you should be aware of this potential source of interference with your users.

D. Updating a Shared Segment

Just when you have the SSERV fire fought, another of your colleagues tells you that he's finally gotten hold of a C compiler for CMS. He's going to install it, but he wants you to make the CMS EDITOR (the old one) understand a filetype of "C", so you sit down and slam out a mod to DMSEDF:

FILE: DMSEDF CFDEF0 A1

```
 ./ I 00740000          $ 740900 900                    11/21/80 16:20:31
          DC    CL8'C',A(C)                                      CFDEF0
 ./ I 01620000          $ 1620200 200                   11/21/80 16:20:31
 C        DS    0F                  C LANGUAGE                   CFDEF0
          DC    C'S',X'00'          MIXED CASE, NO SERIALIZATION CFDEF0
          DC    C'V',AL1(160)       RECFM V, LRECL 160           CFDEF0
          DC    AL1(0,0)            NO TRUNC, VERIFY ALL         CFDEF0
          DC    A(CTABS)            DEFAULT TAB STOPS            CFDEF0
          SPACE 2                                                CFDEF0
 ./ I 01720000          $ 1721000 1000                  11/21/80 16:20:31
 CTABS    DC    AL1(1,4,7,10,13,16,19,22,25,28,31,34,37,40,43,46) CFDEF0
          DC    AL1(49,52,55,58,61,64,67,70,73,76,0)             CFDEF0
```

FILE: DMSEDF AUXLCL A1

```
 * * * * * * * * * * * * * * *  AUXLCL  * * * * * * * * * * * * * * * * *
 CFDEF0 - 05/22/79 - DEFINE FILE DEFAULTS FOR C LANGUAGE PROCESSOR.

 vmfasm dmsedf dmssplcl
 UPDATING 'DMSEDF ASSEMBLE D1'
 APPLYING 'DMSEDF CFDEF0 A1'
 ASMBLING DMSEDF
 ASSEMBLER (XF) DONE
 NO STATEMENTS FLAGGED IN THIS ASSEMBLY
 DMSEDF TEXT CREATED
```

Again, you have the problem of how to install the change. This time, there is no friendly bucket writer to tell you, so you look in Appendix C of the sysgen manual.* You find there a table which tells you what needs to be

---

* Appendix C is not infallible. There have been errors in it, but the real problem is that it doesn't list every CMS module. A more reliable way to decide how to put a given CMS textfile into production is to look for its name in DMSZER ASSEMBLE, DMSSEG ASSEMBLE, the CMSGEND EXEC, and the CMSLOAD EXEC (the CMS nucleus loadlist). If the name is listed in DMSZER ASSEMBLE, then the textfile is part of the CMSZER shared segment, which you build with CMSZGEN. If the name is listed in DMSSEG ASSEMBLE, then the textfile is part of the CMSSEG shared segment, which you build with CMSXGEN. If the textfile is listed in the CMSGEND EXEC, then it must be incorporated into some CMS MODULE via CMSGEND. If the name is listed in the CMS loadlist, then the textfile is part of the CMS nucleus; you install it by rebuilding the nucleus with VMFLOAD. Note, too, that all the textfiles between DMSNUC and the first SLC statement in the CMS loadlist are also part of CMSZER.

regenerated when a given CMS module is changed and even what EXECs to use to do it. You look up DMSEDF in this table and find that you need to regenerate the EDIT MODULE, the EDMAIN MODULE, and the CMSSEG shared segment and that you do these things with CMSGEND and CMSXGEN. You use CMSGEND to generate an EDIT MODULE and an EDMAIN MODULE on your A-disk. One command does them both:

```
 cmsgend edit

 *** RESULTS:
 ' EDIT MODULE A2 ' CREATED FROM TEXT DECK ( S ) DMSEDX DMSEDF DMSZIT
 *** RESULTS:
 ' EDMAIN MODULE A2 ' CREATED FROM TEXT DECK ( S ) DMSEDX DMSEDF
```

Once you have the two editor modules on your A-disk, you can test your mod by just issuing an EDIT command. Like almost all the other CMS MODULEs, EDIT is loaded from a user's disk even if it exists in the shared segment and on the S-disk. If you were applying a fix to XEDIT, however, you couldn't test the new XEDIT MODULE from your A-disk without first getting rid of the CMSSEG shared segment (by re-IPLing with PARM SEG=NULL). The shared segment version of XEDIT is always used in preference to a disk-resident version. Alternatively, you could test a new version of XEDIT by invoking XEDMAIN, rather than XEDIT; the XEDMAIN MODULE is picked up from your disk, rather than from the shared segment. (Note, however, that you can't test EDIT by invoking EDMAIN.) It's not clear why the authors of XEDIT chose to violate CMS conventions in these ways, but they did, so it's something you must remember when you test fixes or mods to XEDIT.

You test your EDIT mod and decide you like it, so you move the new textfile and the two new MODULEs to the S-disk. Then, of course, you do a SAVESYS and a DMSZES to get the saved S-disk directories back in line. That gives you a new EDIT MODULE S2 and a new EDMAIN MODULE S2, but the editor also lives in the CMSSEG shared segment, so you still need to regenerate CMSSEG.

The CMSXGEN EXEC, which is used to generate CMSSEG, is another generally reliable tool which you can treat as a primitive. CMSXGEN knows how to build the shared segment, defaulting the shared segment name to CMSSEG and its address to 100000; you may specify a different name and address if you wish. You should be conscious, however, of one problem with CMSXGEN. It won't warn you if you overflow the segment, so it's a good idea always to check the CMSSEG loadmap to make sure everything fits. Before you invoke CMSXGEN, you must make your virtual machine size large enough that CMSXGEN will be able to load the contents of the shared segment into your virtual memory, at the addresses they will have in the shared segment:

```
spool prt to ipcs
define storage 4m
STORAGE = 04096K
CP ENTERED; DISABLED WAIT PSW '00020000 00000000'

ipl 190 clear
VM/SP CMS - 02/28/82 16:36
Y (19E) R/O
CMSZER SYSTEM NAME 'CMSZER' NOT AVAILABLE.
CMSSEG SYSTEM NAME 'CMSSEG' NOT AVAILABLE.

access 391 a ( noprof
access 190 b/a
cmsxgen 210000 cmsseg
PRT FILE 0003  TO  IPCS     COPY 001   NOHOLD
SYSTEM SAVED
CMSXGEN COMPLETE
```

Those two messages about the segments not being available simply mean that the segments can't be attached to your virtual machine because your virtual machine is too big. That's exactly what you want to have happen, since you want to rebuild CMSSEG. Because your virtual printer is spooled to IPCS, the loadmap for this new segment is sent to that virtual machine to be archived.

E. Updating an IPL Deck

You have barely got the new editor installed when you get a call from Level Two asking you to fixtest a fix for a problem you've been having with DDR running standalone. They read you the fix. You key it in and assemble it. You rename DMKDDR TXTTST to DMKDDR TEXT, since this is one of those CP functions which live on the S-disk. Then you have to figure out how to build a version of DDR that you can run standalone to do the test. The sysgen manual tells you to use the GENERATE EXEC, so you do. It seems to work; at any rate, it builds you a new IPL DDR file on your A-disk. But you type out the beginning of that new IPL file and discover that the comments there don't list your new fix; the S-disk textfile was used, not yours, even though you remembered to rename yours to TEXT. So, you take a look at the GENERATE EXEC and decide that, short of modifying it, there is no way you are going to be able to get GENERATE to build you an IPL file from a textfile on your A-disk. Since this is a fixtest, you have no intention of putting the textfile on your S-disk yet, but it turns out to be very easy to build the IPL file "by hand":

```
copyfile ddr loader s dmkddr text a ipl ddr a ( replace
```

The IPL file is quite simply just a loader with the DMKDDR textfile concatenated to it. (Other IPL files are built with 3CARD LOADER S2.) You punch IPL DDR A to cards or MOVEFILE it to tape, so that you can do your standalone test. Later, when you are satisfied that this is a good fix, you invoke CMSGEND to build a new DDR MODULE; then you copy DMKDDR TEXT, DDR MODULE, and IPL DDR from your A-disk to the S-disk and do a SAVESYS and a DMSZES.

F. Updating the CMS Nucleus

The next CMS problem you encounter requires you to apply a fix to DMSITP. You key in the fix and do the assembly and then consult Appendix C. Appendix C tells you that DMSITP is part of the CMS nucleus. When you change DMSITP, you must rebuild both the nucleus and the CMSZER shared segment. You want to test this fix before you put it into production, but you are using your alternate S-disk for something else just now, so you have the problem of how to test this nucleus code. It can be done very nicely using a small disk (which might even be a TDISK) to hold your test nucleus:

```
 define t3350 590 2
 DASD 590 DEFINED
 format 590 k
 '2' CYLINDERS FORMATTED ON 'K(590)'.

 format 590 k 1 ( recomp
 LABEL  CUU M  STAT CYL TYPE BLKSIZE    FILES
 MNT590 590 K   R/W   1 3350  1024          0
```

The disk on which the CMS nucleus resides must be CMS-formatted and must have room for IPL text at the beginning and for the nucleus at the end. The space for the nucleus at the end is set aside by reducing the minidisk's apparent size with the RECOMP option of FORMAT.

Building a CMS nucleus is very much like building a CP nucleus. VMFLOAD is used to punch the files listed in the CMS loadlist, beginning with DMKLD00E, the VM loader. Then the loader file is IPLed. The loader loads the nucleus and passes control to DMSINIW, which prompts you for information about your new nucleus. The reply to the query about the "system disk address" is "190", because 190 is the place where the CMS user running with this new nucleus (i.e., you) will find the CMS MODULEs, etc. The reply to the query about the "IPL device address" is "590", because that's where you want this nucleus written. The "nucleus cylinder/block address" is the number of the first cylinder or block in the space set aside at the end of the nucleus minidisk, in this case, cylinder 1.

```
access 190 f/a
F (190) R/O
190 ALSO = S-DISK

close punch
close reader
spool reader class n
spool punch to * class n
spool prt to ipcs

vmfload cmsload dmssplcl
SYSTEM LOAD DECK COMPLETE
PUN FILE 8837  TO  MAINT    COPY 001   NOHOLD
R;

order reader 8837

ipl 00c clear
DMSINI606R SYSTEM DISK ADDRESS =
190
DMSINI615R Y-DISK ADDRESS =
19e
DMSINI607R REWRITE THE NUCLEUS ?
yes
DMSINI608R IPL DEVICE ADDRESS =
590
DMSINI609R NUCLEUS CYL/BLK ADDRESS =
1
DMSINI610R ALSO IPL CYL/BLK 0 ?
yes
DMSINI611R VERSION IDENTIFICATION =
test cms system
DMSINI612R INSTALLATION HEADING =
princeton university time-sharing system

TEST CMS SYSTEM
```

With that, the new nucleus is written out onto your 590 and IPLed, so that you can test the fix to DMSITP.

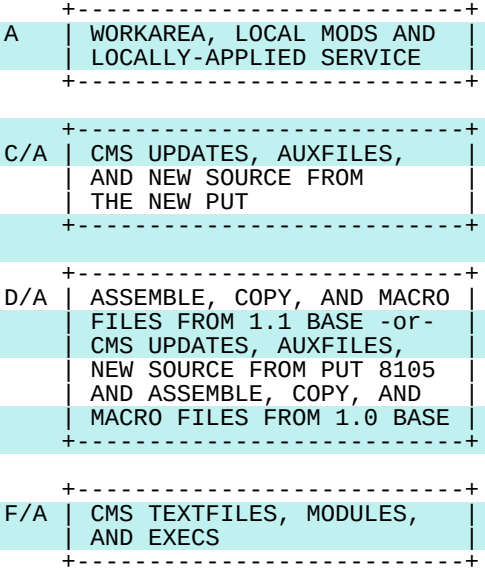
When you are ready to put this fix into production, you copy the new textfile to 190, ACCESS 190 as A, and again invoke VMFLOAD to build a nucleus. This time, you reply "190" to the query about the IPL address. You reply to the query about the nucleus address with the block or cylinder number at which the nucleus starts on your S-disk (the beginning of the area set aside by RECOMP). As soon as the new nucleus is IPLed, you can do a SAVESYS and a CMSZGEN to save the new nucleus and the new CMSZER shared segment. Remember that you must first have made your virtual machine large enough to contain CMSZER. Your virtual printer should again be spooled to IPCS so that the new CMS nucleus loadmap and the new CMSZER loadmap can be archived.

X. Installing Preventive Service for CMS

A. New CMS Service Minidisks

When you are ready to load up the CMS service from a PUT, you first get your alternate S-disk (490) ready to receive the service by using DDR to copy the current S-disk to the alternate. One thing to remember about using DDR to copy S-disks: the size you see when you issue the QUERY DISK S command is the size of the S-disk less the amount of space that has been set aside for the CMS nucleus (by means of the RECOMP option of CMS FORMAT); the command QUERY VIRTUAL 190 will give you the true size; that is the number of cylinders or blocks you want to copy. To be safe, just use DDR COPY ALL.

You may want to swap your 491 with your 391 and your 494 with your 394, but since you can't swap 490 with 190 yet, let's just leave them at those addresses and set up an EXEC to access them like this:



B. Loading the Service

You remember that the layout of the files on the logical service tape for SP was something like this:

1. SP installation EXEC
2. CP auxfiles
3. CP updates (PTFs)
4. CP macro auxfiles
5. CP macro updates
6. New CP source
7. CP maclibs
8. CP textfiles
9. CP loadlist EXECs
10. CMS auxfiles
11. CMS updates
12. CMS macro auxfiles
13. CMS macro updates
14. New CMS source
15. CMS maclibs
16. CMS textfiles
17. Standalone IPL decks
18. LOADER and service EXECs
19. CMS module files
20. HELP files and XEDIT EXECs
21. EREP txtlibs
22. 308x IOCP

You use VMSERV to load the CMS service onto the 490 and 494 disks or you do it by hand with these commands (assuming that your SERVICE DISKMAP shows the SP logical tape starting at file 40 of the PUT):

```
tape rewind
tape fsf 2                  <== skip PUT junk
tape fsf 37                 <== skip VM Release 6 files
tape fsf 1                  <== skip SP installation EXEC
tape fsf 8                  <== skip VM/SP CP files
access 494 c
access 490 f
vmfplc2 load * * c ( eof 5
vmfplc2 load * * f ( eof 6
tape fsf
vmfplc2 load * * f
```

These commands load the new CMS source, the CMS updates, and the CMS auxfiles onto your alternate CMS PUT disk; all other CMS files are loaded onto the alternate S-disk. Wait to load the EREP file to your EREP disk some other day, when you're feeling strong and adventurous. Load the IOCP file only if you are supporting a 308x.

C. Rebuilding the Assembler Auxiliary Directory

Having built the new S-disk from the old S-disk and the files on the service tape, you must rebuild the auxiliary directory for ASSEMBLE. CMSGEND knows how to do the GENDIRT for ASSEMBLE, so you just enter:

```
access 490 a
cmsgend assemble
```

You may have noticed that there is an ASMGEND command. You could use that, instead of CMSGEND, to rebuild the assembler auxiliary directory, but that would be overkill. ASMGEND also rebuilds all the assembler MODULE files from the textfiles. You need to use ASMGEND only if you have applied a mod or a fix to one of the IFNxxxxx textfiles, which doesn't happen often.

D. Building a Nucleus on the Alternate S-Disk

Having loaded up the service, you again go through the steps that we went through for CP earlier. You take the corrective measures the IBM and VMSHARE buckets recommend; you carry forward any old corrective service you still need; and you reapply your local mods. You do the VMFMACs with the LCLMAC control file and the VMFASMs with the DMSSPLCL control file. When you have all the updated textfiles, etc., on your alternate S-disk, you can build a new nucleus on the alternate S-disk:

```
access 490 a
close pun
close rdr
spool pun to * class n
spool rdr class n

vmfload cmsload dmssplcl
PUN FILE 1301   TO  MAINT    COPY 001    NOHOLD
order rdr 1301

system clear
cp detach 190
cp define 490 190

ipl 00c clear
```

VMFLOAD punches a CMS nucleus loader file to your virtual reader. You get rid of the regular S-disk at 190 and the CMS system you were running under. You redefine your alternate S-disk to have a virtual address of 190, because that's where you want it to be when you test this system. You then IPL the CMS loader file and reply to the questions from DMSINI. Answer the ones about the system disk address and the IPL address with "190", not "490". (Your alternate S-disk will be at 190 when it's being used, and that's where it is now, so that's where you want the nucleus written.) It's a good idea to reply to the request for the "version identification" with something like "TEST CMS SP107 mm/dd/yy" to make it obvious that this is a test system. When you've answered all the questions, you have a system for testing the new PUT built on your alternate S-disk.

E. Building Alternate Saved Systems

You also need to build some saved systems for the new PUT, to test it properly. Don't skip this step. It is very often the case that shared-segment code works fine until it's actually put into a shared segment. In DMKSNT, in addition to your regular definitions for CMS, CMSSEG, and CMSZER, create definitions for NEWCMS, NEWSEG, and NEWZER. These new definitions will be very like your standard definitions. Their SYSNAMEs will be different, of course. And you must allocate different DASD locations for these saved systems and describe the locations in the SYSVOL and SYSSTRT parameters. In the NEWCMS definition, VSYSADR should be 190, but VSYSRES and SYSCYL should point to the 490 S-disk.

Then, once you've got that version of DMKSNT in production and you have the new PUT loaded onto 490, you can build your new saved systems:

```
define storage 16m
define 490 190

ipl 190 parm seg=newsegBB zer=newzerBB     ("B" = blank)
savesys newcms

access 491 a ( noprof
access 190 b/a
cmsxgen nnnnnn newseg
cmszgen nnnnnn newzer
```

CMS uses the CMSSEG and CMSZER shared segments by default, so you need to do something to make this NEWCMS system know to use NEWSEG and NEWZER, instead. The PARM on the IPL statement is one way of doing this. When you specify the IPL PARM field as shown here, with the segment names right-padded with blanks, the names "NEWSEG " and "NEWZER " get moved into the SYSNAMES table in the CMS nucleus by the CMS initialization routine before the SAVESYS is done, so the saved NEWCMS system will default to using NEWSEG and NEWZER. This will not be sufficient, however, if anyone is going to be using this system by IPLing it from its virtual address, rather than IPLing the saved system. In that case, you need to change the default shared segment names in your test S-disk-resident nucleus, not just in the saved system. I know people who do this by simply editing DMSNUC TEXT, changing "CMSSEG" and "CMSZER" to "NEWSEG" and "NEWZER", before doing the VMFLOAD to build the nucleus. But, since that seems a little impure, you could build yourself a test version of the SYSNAMES macro and a DMSNUC TXTTST which uses the test macro and a control file that knows about TXTTST textfiles. Incidentally, if you ever have any doubts about what shared segments you are using, you can find out by issuing the QUERY SYSNAMES command.

F. Making the Test CMS System Available to Your Users

You now have a complete test system. Announce to your users that they can try it out by invoking the NEWCMS EXEC, which you've put on the Y-disk:

FILE: NEWCMS EXEC Y2

```
CP DETACH 190
CP LINK MAINT 490 190 RR
CP IPL NEWCMS
```

Be sure to warn them not to invoke this EXEC from their PROFILE EXECs. They're likely to become confused when their PROFILE does an IPL which does a PROFILE which does an IPL, etc.

G. Putting a New Version of CMS Into Production

We have two possible scenarios for making the test CMS the production CMS. The easiest way depends on your having been able to position your 490 disk in a good place as far as performance is concerned (middle of volume, etc.). In that case, you just want to swap the two S-disks. Here are the steps involved in that:

- Rebuild DMKSNT, changing the SYSNAMES for CMS, CMSSEG, and CMSZER to OLDCMS, OLDSEG, and OLDZER, and changing NEWCMS, NEWSEG, and NEWZER to CMS, CMSSEG, and CMSZER.
- Rebuild the CMS nucleus on 490 as before, but specify a "version identification" that will be suitable for production or use the default.
- Swap the directory entries for 190 and 490.
- Install the new directory just before shutting CP down.
- IPL the new CP nucleus containing the new DMKSNT.
- Before letting any real users log on, resave your new CMS system without specifying the segment names, so that the defaults will be used:

```
ipl 190
savesys cms
```

The other scenario is for putting the new CMS into production if you can't afford to have both S-disks in good places. In that case, you'll want to copy 490 to 190, rather than swapping the two. For this, you need some dedicated test time. First, back up your 190 (DDR to tape). Next, DDR COPY 490 to 190 and:

```
define storage 16m
ipl 190
savesys cms
access 391 a
access 190 b/a
cmsxgen nnnnnn cmsseg
cmszgen nnnnnn cmszer
```

Then, restore the backup of the old system to 490, just in case you have a disaster ahead of you. (It's so nice to have at least one truly runnable CMS system online at all times.) Finally, open it up to the users.

XI. Converting to a New Release of CMS

When you are ready to go to a new release of CMS, get yourself a new S-disk at address, say, 590 and format it and RECOMP it. Then, you are ready to load up the files from the distribution tape. All the CMS files from the distribution tape go onto the new S-disk, except the ASSEMBLE, COPY, and MACRO files. So, you load the CMS textfiles, MODULEs, EXECs, maclibs, IPL files, etc., onto the new S-disk and the source files onto a new base disk.

Note that until you get your new CMS built and can run under it, you need to access 590 after the S-disk for your current system as, say, mode W; if the new S-disk were accessed before the production S-disk, you would end up using CMS MODULEs and textfiles from the new release, which are very likely to be incompatible with the CMS that you are running under. However, if there are files on the new S-disk that you need to use while you are building your new system, these must be accessed ahead of the corresponding files on the production S-disk. This complication is another reason why the purists feel that many of the files that are on the S-disk should really be on service disks instead. Probably the easiest way to handle this difficulty is to access the new S-disk as a whole after your production S-disk and to access components of the new S-disk in front of the production S-disk:

```
access 590 w/a
access 590 f/a * exec f1
access 590 g/a * maclib g2
```

If you have a service tape to be applied atop the new base, you load it just as we did before, and, again, you go through the steps of applying the service recommended in the error bucket, reapplying any old corrective service you still need, and reapplying your local mods. Remember also to re-CMSGEND the assembler. Having done all that, you have a complete S-disk and can build a new CMS nucleus on that S-disk using VMFLOAD. You should also build new test saved systems to use to test this new release before you put it into production. Check the sysgen manual for the new release to see what the saved systems for the new release should look like and put appropriate new entries into your DMKSNT.

Again, the process of installing a new release is very much the same as the process of putting a new service level into production.

XII. Installing Service on Program Products

Many of the VM program products, such as PASSTHRU, for example, are installed and maintained in very much the same way that the SCP itself is installed and maintained. Once you understand the philosophy of installing service on CP and CMS, you should have no trouble applying service to these products as long as they have been packaged correctly. Most of these products look more like CMS than like CP; you may have to generate MODULEs for them and put these files on the Y-disk. You may find that they don't have nice EXECs like CMSGEND to generate the MODULEs for you, but, if they don't, there will be instructions somewhere in their manuals telling you the commands you need to regenerate any MODULEs. You will need to check most of the program product installation and service EXECs to see whether they reference the CP or CMS maclibs, in which case you may have to modify the EXECs to reference the correct maclib names for your installation and the version of VM that you are running.

XIII. Converting from SP1 CMS to SP2 CMS

Converting from SP1 CMS to SP2 CMS turns out to be remarkably easy. If you define the saved systems and set up the minidisks before your SP2 tape arrives, you can have the new CMS up and running within ten minutes of getting the tape. I am very pleased to be able to report that IBM put some effort into making this new CMS run under an old CP; for example, the new CMS IDENTIFY command checks to see whether an SP2 CP command it needs is available and, if it isn't, uses an older CP DIAGNOSE instead. Most of SP2 CMS seems to function properly under SP1 CP, with one exception: if you have applied APAR VM14879 to your Release 1 CP system, you will be unable to IPL an SP2 saved CMS system. However, you will have the same problem if you have applied VM14879 to your Release 2 CP system.

I'll be going through the steps for bringing up SP2 CMS, using the assumptions that you are running SP1 CP and CMS and that you want to bring the new CMS up alongside the old one, rather than replacing it immediately.

A. SP2 Changes that Affect CMS Installation and Service

There are some structural changes in SP2 CMS which you need to be aware of before you try installing it.

First, the HELP files no longer default to being on the S-disk, although they can be, if you wish. By default, they are on a separate disk at virtual address 19D, as mode 1 files, except that the HELPMENU files are mode 2. As far as I can see, making the MENU files mode 2 is not magic; they did it that way just because they wanted the disk to appear to contain a few files when HELP accesses only the mode 2 files. You may want to consider moving your local HELP files onto 19D, but the HELP command can use HELP files on other disks, so you don't have to move your files. However, HELP can no longer find mode 1 HELP files on the Y-disk, so, if you have any such files, you must move them, change their mode, or modify HELP. HELP accesses the HELP files at the first vacant disk mode. This may cause you problems if you are expecting local HELP files on a disk later in the search order to be used in preference to IBM's files of the same name. We have tentatively decided to move our local HELP files to the 19D in order to get them off of the Y-disk (and out of the YSTAT). One thing you should keep in mind that the sysgen manual neglects to mention is that you need to add DIRECTORY LINKs to 19D for all your CMS users; this may cause problems for users who are already using 19D for something else. Another change in HELP that you should be aware of is that the format of the HELP screen has been changed; there are now two fewer lines of data on the screen than formerly. So, if your local HELP files have been carefully formatted to look pretty on the screen, they won't anymore.

Second, IBM has now decided that EREP should be put back onto the S-disk. I disagree. EREP is a separate component and causes plenty of headaches in its own right. You shouldn't have to cope with new EREP bugs at the same time you are coping with the bugs in a new level of CMS. And, it is ridiculous to have to resave your CMS systems just to apply an EREP fix. Besides, the EREP libraries are huge and will make two big, practically-unused holes in the middle of your S-disk. When you decide to risk going to a new EREP, I recommend that you load up the current EREP txtlibs and the current EREP MODULE onto a 201 disk and run EREP from there.

Third, there is no longer a CMSZER shared segment. The sharable nucleus code and the SSTAT and YSTAT (the directories for the S-disk and Y-disk), which used to be in CMSZER, have been folded back into the CMS saved system.

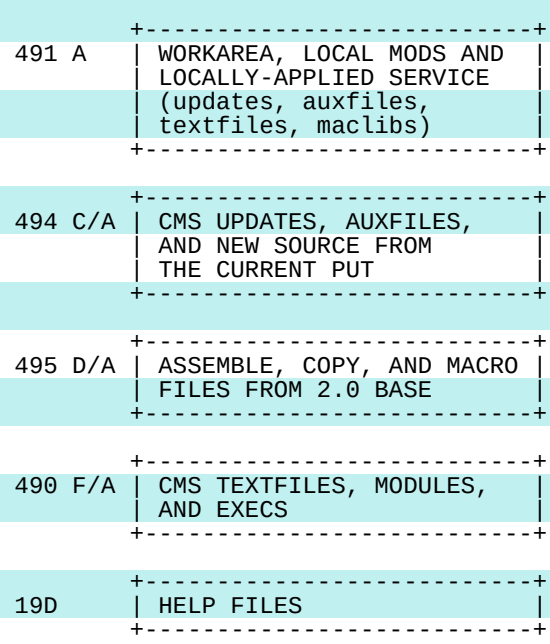
Fourth, the CMS nucleus is now loaded at a high address, rather than being loaded into segments 0 and 1. One result of this is that you may find that you cannot IPL 190 because your virtual machine size isn't large enough to contain the nucleus. Another result of moving the nucleus to a higher address is that it ends up in the middle of larger virtual machines. It will work that way, but the big user's free storage is fragmented into one piece below the nucleus and another above it. Furthermore, the piece above the nucleus cannot be used by CMS OS GETMAIN simulation, no matter how big you make the virtual machine. Some installations have decided to get around this problem by generating their CMS nucleus up in the sixteenth megabyte of virtual memory. But this means that all their CMS users will have segment tables for 16 megs of segments; moving CMS from its default location to the sixteenth meg increases the CP free storage required for segment tables by 112 doublewords for each CMS user. IBM's recommendation is to build two CMS saved systems, "CMS" and "CMSL", the latter being generated at a much higher address than the former. (They provide two CMS loadlist EXECs, CMSLOAD and CMSLOADL, for building these two CMS systems.) But if you do that, you may end up with twice as many shared pages in real memory as you have now. You'll have to decide what is best for you. One approach being taken by many installations is to install a saved system named "CMS" which is not really CMS, but is, instead, an IPLable program which decides which CMS the user needs and IPLs it for him. That way, the general user need never be aware of the two sizes of CMS systems. Obviously, IBM should have provided this function when it decided to give us two flavors of CMS. But, since IBM didn't, Cornell University did. Appendix E contains Larry Brenner's program to build such an IPLable system.

Fifth, some benighted soul decided that in SP2 the CP maclib, DMKSP MACLIB, should be on the S-disk, rather than on a CP service disk. When you build your S-disk from the distribution tape, DMKSP MACLIB will end up on the S-disk. This is madness, of course, so you should copy the maclib to your CP base service disk and erase it from the S-disk. After a considerable struggle, IBM has accepted APAR VM16999 against this problem. The first SP2 PUT will move the CP maclib off the S-disk onto the 194.

B. Before the Tape Arrives

It can be very confusing juggling two releases of CMS. For example, I've been told by others that the assembler goes crazy if you accidentally mix the nucleus and the S-disk from SP1 and SP2. In general, I find life to be easier if I can start running under a new CMS immediately, rather than switching back and forth between two flavors of CMS or trying to do assemblies for one level of CMS under another level. For that reason, I suggest that you first build a totally unmodified SP2 CMS and later apply your mods and corrective service to it once you can run under it. Of course, I'm not suggesting that you open it up to the world in that condition.

You can get your new CMS service minidisks set up before your tape arrives. I recommend a service disk layout along the lines of the following:



This layout includes an A-disk, an S-disk, a HELP disk, and source disks for the base and PUT. If your 491 and 494 from the earlier layout are available, you can use them for the A-disk and the PUT source disk. The SP2 base CMS source fits nicely in 55 cylinders of 3350. The new S-disk can be quite a bit smaller than the SP1 S-disk was (assuming you aren't planning to put EREP on it). Thirty 3350 cylinders will give you an S-disk which holds vanilla SP2 CMS and two one-cylinder nuclei and is only 85 percent full. Fifteen 3350 cylinders for the 19D holds all of IBM's HELP files and our local HELP files and leaves plenty of room for growth. When you are calculating the sizes for your minidisks, note that the CMS nucleus

has grown quite a bit, to 46 pages (368 FBA blocks), so be sure to allocate enough room for it. In fact, I recommend that you set aside room for two CMS nuclei on your new S-disk, not for backout purposes, but because you are going to want to be able to IPL both the large and the small CMS nucleus by address, as when you want to resave your CMS systems because of some change to the S-disk or Y-disk. Format your new minidisks and build a PROFILE EXEC for your new A-disk which will access the service disks in the order shown. Remember to RECOMP the S-disk, to leave room for the nuclei. (In my case, the 30-cylinder S-disk is RECOMPed to 28 cylinders, leaving room for two 1-cylinder nuclei, in cylinders 28 and 29.)

Once you've defined the SP2 minidisks, you can set up the new saved system definitions in DMKSNT. You may want to set everything up with default values to begin with, so that you can get a Release 2 CMS built right away. The Release 2 sysgen manual will tell you what the saved system definitions should look like, except that it forgets to tell you about the CMSSEG definition for the "large" CMS nucleus. (The sample DMKSNT files on the distribution tape don't have it either, but the Program Directory does.) Here are the default definitions I used for a system with 3350's:

```
*                                                               VMSPR2
*  DMSALPHA AT X'1D0000'                                        VMSPR2
SP2CMS   NAMESYS  SYSNAME=SP2CMS,SYSSIZE=256K,                  VMSPR2*
                  VSYSADR=190,VSYSRES=XXX974,SYSCYL=031,        VMSPR2*
                  SYSVOL=VMM018,SYSSTRT=(262,1),SYSPGCT=73,     VMSPR2*
                  SYSPGNM=(0-4,14-33,464-511),SYSHRSG=(29,30,31)VMSPR2
*                                                               VMSPR2
*  SP2SEG AT X'1A0000'                                          VMSPR2
SP2SEG   NAMESYS  SYSNAME=SP2SEG,SYSSIZE=96K,                   VMSPR2*
                  VSYSADR=IGNORE,SYSCYL=,VSYSRES=,              VMSPR2*
                  SYSVOL=VMM010,SYSSTRT=(223,60),SYSPGCT=48,    VMSPR2*
                  SYSPGNM=(416-463),SYSHRSG=(26,27,28)          VMSPR2
*                                                               VMSPR2
*  DMSALPHA AT X'F00000'                                        VMSPR2
SP2CMSL  NAMESYS  SYSNAME=SP2CMSL,SYSSIZE=256K,                 VMSPR2*
                  VSYSADR=190,VSYSRES=XXX974,SYSCYL=031,        VMSPR2*
                  SYSVOL=VMR001,SYSSTRT=(327,1),SYSPGCT=73,     VMSPR2*
                  SYSPGNM=(0-4,14-33,3840-3887),                VMSPR2*
                  SYSHRSG=(240,241,242)                         VMSPR2
*                                                               VMSPR2
*  SP2SEGL AT X'ED0000'                                         VMSPR2
SP2SEGL  NAMESYS  SYSNAME=SP2SEGL,SYSSIZE=96K,                  VMSPR2*
                  VSYSADR=IGNORE,SYSCYL=,VSYSRES=,              VMSPR2*
                  SYSVOL=VMM018,SYSSTRT=(263,1),SYSPGCT=48,     VMSPR2*
                  SYSPGNM=(3792-3839),SYSHRSG=(237,238,239)     VMSPR2
```

These entries define a "small CMS" named SP2CMS and a "large CMS" named SP2CMSL, with CMSSEG shared segments named SP2SEG and SP2SEGL. These names are used rather than "CMS" and "CMSSEG" because we are still using those for our production SP1 CMS system. Note that the saved CMS system needs quite a bit more DASD space than before. Once you've figured out how much space you need for these additional saved systems, find the space on your system-owned volumes, CP-format it, add the new saved system definitions to your DMKSNT, and install a new CP nucleus containing the new DMKSNT.

C. Loading up SP2 CMS from the Distribution Tape and the PUT

Once the saved system spaces are formatted, the new DMKSNT is installed, and the minidisks are allocated and formatted, you've done all you can until the tape arrives. When it comes, you are ready to load it up. The files on the distribution tape are in the following order:

1. Installation EXEC
2. Sample files
3. CP textfiles
4. HELP files
5. EREP
6. CMS textfiles, EXECs, maclibs
7. CP source
8. CMS source

(The last two items are on separate reels, if you get 1600 bpi tapes.) There is an elaborate new installation EXEC, PREP EXEC, on the distribution tape. I loaded it up, but it looked like an awful lot of bother, so I just erased it and did everything "by hand." That's the approach I will be demonstrating.

First, access your new disks in write mode. Note again that until you get your new CMS built and can run under it, you need to access 490 after the S-disk for your current system as, say, mode Z.

```
access 19d b                <== HELP disk
access 495 d                <== base source disk
access 490 z                <== new S-disk
```

Mount the tape on 181 and load it up, skipping EREP and the CP stuff and erasing the CP maclib which gets loaded onto the S-disk:

```
tape fsf 3
vmfplc2 load * * b
tape fsf
vmfplc2 load * * z
erase dmksp maclib z
tape fsf
vmfplc2 load * * d
```

If you've been able to get hold of an SP2 PUT, then you should load it next. The file layout in the SP2 "logical tape" on the VM PUT is very similar to the SP1 file layout, but with a few differences:

1. SP installation EXEC
2. CP auxfiles
3. CP updates (PTFs)
4. CP macro auxfiles
5. CP macro updates
6. New CP source
7. CP maclibs
8. CP textfiles
9. CP loadlist EXECs
10. CMS auxfiles
11. CMS updates
12. CMS macro auxfiles
13. CMS macro updates
14. New CMS source
15. CMS maclibs
16. CMS textfiles
17. Standalone IPL decks
18. LOADER, service EXECs, XEDIT EXECs
19. CMS module files
20. EREP txtlibs
21. HELP files
22. 308x IOCP

The differences are that the XEDIT macros are now in the file with the service EXECs, rather than with the HELP files, and files 20 and 21 have been interchanged. To apply the CMS portion of the SP2 PUT, position the tape to the beginning of the SP2 logical tape and VMFPLC2 LOAD files 10-14 to your new PUT source disk (494), files 15-19 (and possibly 22) to your new S-disk (490), and file 21 to your new HELP disk (19D). By the way, despite what the sysgen manual says, the filetypes of SP2 APAR fixes have a prefix of "L", except that the CMS macro APARs have a prefix of "Z".

D. Building the CMS Nuclei and Saved Systems

Now you are ready to build a CMS nucleus, just as we've done before:

```
spool prt to ipcs
spool pun to * class n
spool rdr class n
access 490 a
vmfload cmsload dmssp
SYSTEM LOAD DECK COMPLETE

define storage 16m
CP ENTERED

detach 190
DASD 190 DETACHED
define 490 190
DASD 190 DEFINED

ipl 00c clear

DMSINI606R SYSTEM DISK ADDRESS = 190
DMSINI615R Y-DISK ADDRESS = 19e
DMSINI640R HELP DISK ADDRESS = 19d
DMSINI604R REWRITE THE NUCLEUS ? yes
DMSINI608R IPL DEVICE ADDRESS = 190
DMSINI082E IPL DEVICE ERROR - REENTER
#cp link * 190 190 mr
DMSINI608R IPL DEVICE ADDRESS = 190
DMSINI609R NUCLEUS CYL/BLK ADDRESS = 29
DMSINI610R ALSO IPL CYL/BLK 0 ? yes
DMSINI611R VERSION IDENTIFICATION = sp2 cms
DMSINI612R INSTALLATION HEADING = princeton university time-sharing system

SP2 CMS
```

(If you get that "IPL DEVICE ERROR" message, you need to establish a write link to 190.) The only thing new here is the prompt for the address of the HELP disk, normally 19D. You can specify some other address for the HELP disk, if you prefer. You can even put the HELP files on the S-disk, as in SP1, in which case you would reply "190" to DMSINI640R.

At this stage, you've got a runnable CMS system and should never have to go back to Release 1, so you might find it easier to define this new S-disk as 190 in your DIRECTORY entry, leaving the current S-disk as 190 for everyone else, of course. The rest of the presentation will assume that the SP2 S-disk is at address 190.

There are a couple of things which still need to be done to the new S-disk. The first is the one I always forget, the regeneration of the assembler's auxiliary directory:

```
 access 190 a
 cmsgend assemble
 ENTER GENDIRT TARGET DISK MODE LETTER
 s
 ASSEMBLE MODULE A2 CREATED FROM DMSASM DMSASD
 access 491 a
 access 190 f/a
```

You specify the "target disk mode" as "S", because that's where the ASSEMBLE MODULE will be when it's being used.

The other thing you need to do is new with this release and not documented in the sysgen manual -- you must build a SYSTEM NETID file to define your systems to the new IDENTIFY command. There is a sample SYSTEM NETID on the distribution tape, and it should now be on your new S-disk:

FILE: SYSTEM NETID S2

```
+-----------------------+
|  *CPUID NODEID NETID  |
|  00000  NODEID NETID  |
+-----------------------+
```

You need to update the SYSTEM NETID file with the CPU serial numbers, node names, and RSCS virtual machine names of all your CPUs. The sample file from the tape is in error in showing a five-character CPU serial number; IDENTIFY won't work unless you give it six-character serial numbers.

I'm going to dig my heels in and insist that this file should be maintained with UPDATE, so it's time to build local control files for the new CMS:

FILE: DMSSPLCL CNTRL

```
 TEXT MACS DSLCLMAC DMSSP CMSLIB OSMACRO DOSMACRO TSOMAC DMKSP
 TEXT AUXLCL
 TEXT AUXPTFS
 TEXT AUXSP
```

To build DMSSPLCL CNTRL, start with the DMSSP CNTRL from your new S-disk. Note that there are no longer 'AUXSP11' and 'AUXSP12' entries, but there will soon be an 'AUXSP21' entry, alas. Note, too, that you still need a separate LCLMAC CNTRL, since the developers haven't done anything about DMSERR and DMSABN being the names of both macros and ASSEMBLE files.

Build an update file to sequence SYSTEM NETID and another update file to define your configuration and an auxfile which lists both these updates:

FILE: SYSTEM SEQUENCE

```
./  S
```

FILE: SYSTEM SYSGEN

```
./ R  00002000 00002000 $ 2100 100
020470 PUCC3081 VMRSCS
220470 PUCC3081 VMRSCS
010577 MAE4341  VMRSCS
014099 ASIS4331 VMRSCS
```

FILE: SYSTEM AUXLCL

```
*$*$*$*$*$*$*$*$*$*$*$*$*$*$*$ AUXLCL $*$*$*$*$*$*$*$*$*$*$*$*$*$*$*$*
SYSGEN - 08/07/82 - PRINCETON CPU ID'S, NODE NAMES, RSCS VM NAMES
SEQUENCE - 08/07/82 - SEQUENCE SYSTEM NETID, SINCE IBM WON'T
```

You will be replacing the SYSTEM NETID file that came on the S-disk with an updated version, so copy the original to your base source disk, so you will always have a copy to update. Then build an updated copy of SYSTEM NETID on your A-disk and copy it to the S-disk, remembering to make it mode 2:

```
access 190 f/a
access 495 d
copy system netid f = = d ( olddate
access 495 d/a
update system netid d dmssplcl ( ctl rep
access 190 f
copy system netid a system netid f2 ( olddate replace
access 190 f/a
```

That gets the S-disk in order. You can now build a "large CMS" nucleus:

```
spool prt to ipcs
spool pun to * class n
spool rdr class n

access 190 a
vmfload cmsloadl dmssp
SYSTEM LOAD DECK COMPLETE

define storage 16m
ipl 00c clear
DMSINI606R SYSTEM DISK ADDRESS = 190
DMSINI615R Y-DISK ADDRESS = 19e
DMSINI640R HELP DISK ADDRESS = 19d
DMSINI604R REWRITE THE NUCLEUS ? yes
DMSINI608R IPL DEVICE ADDRESS = 190
DMSINI609R NUCLEUS CYL/BLK ADDRESS = 28
DMSINI610R ALSO IPL CYL/BLK 0 ? no
DMSINI611R VERSION IDENTIFICATION = sp2 large cms
DMSINI612R INSTALLATION HEADING = princeton university time-sharing system
```

This is just like building the small nucleus, except for four things:

1. The CMSLOADL loadlist is used in the VMFLOAD command;
2. The nucleus is written on the other nucleus cylinder;
3. The cylinder-zero IPL text is not rewritten; and
4. A different version identifier is specified.

Incidentally, instead of keeping both CMS nuclei on the S-disk, you might prefer to keep one on the S-disk and one on the Y-disk. That way, you could have cylinder-zero IPL text for both of them. To do this, you would have to RECOMP room for a nucleus at the end of your Y-disk. Then, you would reply "19E" to DMSINI608R and "yes" to 610R, and you would specify the correct Y-disk cylinder number in response to 609R.

Having built both nuclei somewhere, you can build the new saved systems, SP2CMS and SP2CMSL:

```
 define storage 16m

 ipl 190 clear parm seg=sp2segBB          ("B" = blank)
 SP2 CMS
 savesys sp2cms
 SYSTEM SAVED
 SP2 CMS

 ipl 190 28 clear parm seg=sp2seglB  <or>  ipl 19e clear parm seg=sp2seglB
 SP2 LARGE CMS
 savesys sp2cmsl
 SYSTEM SAVED
 SP2 LARGE CMS
```

Note that IPLing 190 IPLs the cylinder-zero IPL text, which brings in the small nucleus from cylinder 29. To IPL the large nucleus from 190, you simply specify its cylinder (28) in the IPL command. You specify segment names in the IPL commands so that those will become the default segment names in the saved systems you are building. Be sure to remember to right-pad the segment names with blanks to eight characters.

Now, you can invoke CMSXGEN to build your shared segments, the CMSSEGs for the two new saved systems:

```
 spool prt to ipcs
 define storage 16m

 ipl sp2cmsl parm seg=null
 SP2 LARGE CMS
 access 491 a ( noprof
 access 190 b/a
 cmsxgen 1a0000 sp2seg
 SYSTEM SAVED
 CMSXGEN COMPLETE
 cmsxgen ed0000 sp2segl
 SYSTEM SAVED
 CMSXGEN COMPLETE
```

You specify NOPROFILE when you access your A-disk because your profile might access unneeded disks and use memory needed for saving the shared segments. You access your S-disk read-only so that the segment maps won't be written on it and obsolete the SSTATs in your newly-saved systems. If you want the maps on the S-disk, then do the SAVESYSs for the two CMS systems after you have built the shared segments.

E. What To Do If You Don't Like the Default Shared Segment Locations

At Princeton, we've tentatively decided to put the small CMS and its shared segment high enough (5 megs) that they will almost always be the only ones in use and to move the shared segment for the large CMS up a bit into the sixteenth meg, so that the big users can have the bottom 15 megs to themselves. We also want to expand the nucleus area allocated to the shared SSTAT and YSTAT by one segment, so that we can use Brown's idea of adding a shared RSTAT for the HELP files.

Moving the shared segments is no problem. As in the past, you need only change the definitions in DMKSNT and specify the new, non-default addresses in the CMSXGEN commands. These are our new shared segment definitions:

```
*                                                               VMSPR2
*  SP2SEG AT X'500000'                                          VMSPR2
SP2SEG   NAMESYS  SYSNAME=SP2SEG,SYSSIZE=96K,                   VMSPR2*
                  VSYSADR=IGNORE,SYSCYL=,VSYSRES=,              VMSPR2*
                  SYSVOL=VMM010,SYSSTRT=(223,60),SYSPGCT=48,    VMSPR2*
                  SYSPGNM=(1280-1327),SYSHRSG=(80,81,82)        VMSPR2
*                                                               VMSPR2
*  SP2SEGL AT X'F40000'                                         VMSPR2
SP2SEGL  NAMESYS  SYSNAME=SP2SEGL,SYSSIZE=96K,                  VMSPR2*
                  VSYSADR=IGNORE,SYSCYL=,VSYSRES=,              VMSPR2*
                  SYSVOL=VMM018,SYSSTRT=(263,1),SYSPGCT=48,     VMSPR2*
                  SYSPGNM=(3904-3951),SYSHRSG=(244,245,246)     VMSPR2
```

The SP2 Program Directory says that, "The first 64K segment after the CMS saved system is reserved for CMS's use." It warns you to leave that segment empty, rather than allocating it to a shared segment. But, if you are virtual memory constrained, you may not find it acceptable to lose that segment. The basis for the warning is that CMSXGEN (and similar EXECs) cannot build a shared segment immediately above the running CMS nucleus. When CMSXGEN tries to load the contents of the segment into virtual storage before doing the SAVESYS for the shared segment, it gets the error message DMSLIO109S VIRTUAL STORAGE CAPACITY EXCEEDED. This is because CMSXGEN loads into free storage without first allocating that storage -- a highly dubious technique. Fortunately, DMSLDR won't allow the free storage chain element right above the nucleus to be overlaid, so it issues the error message and quits. But, if you can contrive to build a segment immediately above the nucleus, CMS is perfectly happy to use it. The obvious way to do this is to build the shared segment while running some CMS nucleus other than the one the segment abuts. I have tried this and it works, so these are the commands for rebuilding the shared segments at their new addresses:

```
define storage 16m
ipl sp2cms
access 491 a ( noprof
access 190 b/a
cmsxgen 500000 sp2seg
cmsxgen f40000 sp2segl
```

You should be aware of a mysterious warning in the Program Directory which says that due to a bug in DMSLDR you must not save shared segments anyplace above the location of the nucleus you are running from. I have tried this only with the CMSSEG for my large CMS nucleus, and I had no trouble with that, but apparently there is a sporadic problem with other segments, such as APL or GDDM. The fix is VM16182.

F. What To Do If You Don't Like the Default Nucleus Location or Size

Moving the CMS nucleus is not so easy as moving the shared segment. The locations of the various parts of the CMS nucleus in SP2 are determined by LOADER SLC ("set location counter") statements inserted here and there in the loadlist EXEC. That's why there are different loadlists for the large and small nuclei -- the SLCs are different. The address of the start of the main portion of the nucleus is determined by an SLC entry in the loadlist just before the entry for DMSALP ("ALP" is short for "alpha"). This is the part of the small CMS loadlist (CMSLOAD EXEC) that defines the location of DMSALP:

```
+-------------------------------------------------------------+
|    .                                                        |
|    .                                                        |
|  &1 &2 &3 DMSINI                                            |
|  ******** DMSZIT MUST BE THE LAST MODULE BEFORE DMSALP      |
|  &1 &2 &3 DMSZIT                                            |
|  &1 &2 &3 SLC L1D0000                                       |
|  ******** DMSALP MUST BE THE FIRST MODULE IN THE NUCLEUS    |
|  &1 &2 &3 DMSALP                                            |
|  &1 &2 &3 DMSCAT                                            |
|    .                                                        |
|    .                                                        |
+-------------------------------------------------------------+
```

Actually, the loadlist record that says "SLC L1D0000" is not an SLC statement itself; it's the name of a CMS file which contains an SLC statement:

FILE: SLC L1D0000 S1

```
+----------------+
|  bSLC  1D0000  |                    ("b" = X'02')
+----------------+
```

If you want the small CMS nucleus to start somewhere other than at 1D0000, you must change that entry to point to a new SLC file which you've built and put on the S-disk. You must also make a corresponding change to the SLC entry just before DMSOME ("omega"), which marks the end of the nucleus:

```
        +-----------------------------------------------------------+
        |    .                                                      |
        |    .                                                      |
        |  &1 &2 &3 DMSVSR                                          |
        |  ******** DMSSIG MARKS THE END OF EXECUTABLE CODE         |
        |  &1 &2 &3 DMSSIG                                          |
        |  &1 &2 &3 SLC L200000                                     |
        |  ******** DMSOME MARKS THE END OF THE NUCLEUS             |
        |  &1 &2 &3 DMSOME                                          |
        |  &1 &2 &3 LDT STARTADR                                    |
        +-----------------------------------------------------------+
```

Expanding the CMS nucleus to make more room for the shared SSTAT and YSTAT also involves modifying the CMS loadlists. The shared SSTAT and YSTAT are now stored between DMSSIG ("sigma") and DMSOME in the CMS nucleus. To expand the space available for the SSTAT and YSTAT, you increment that last SLC by one segment. You must also, of course, modify your DMKSNT entry for the saved system to show that it contains the additional segment.

The modified DMKSNT entries for our two saved CMS systems are as follows:

```
  *                                                                VMSPR2
  *  DMSALPHA AT X'530000'                                         VMSPR2
  SP2CMS   NAMESYS  SYSNAME=SP2CMS,SYSSIZE=256K,                   VMSPR2*
                 VSYSADR=190,VSYSRES=XXX974,SYSCYL=031,            VMSPR2*
                 SYSVOL=VMM018,SYSSTRT=(262,1),SYSPGCT=89,         VMSPR2*
                 SYSPGNM=(0-4,14-33,1328-1391),                    VMSPR2*
                 SYSHRSG=(83,84,85,86)                             VMSPR2
  *                                                                VMSPR2
  *  DMSALPHA AT X'F00000'                                         VMSPR2
  SP2CMSL  NAMESYS  SYSNAME=SP2CMSL,SYSSIZE=256K,                  VMSPR2*
                 VSYSADR=190,VSYSRES=XXX974,SYSCYL=031,            VMSPR2*
                 SYSVOL=VMR001,SYSSTRT=(327,1),SYSPGCT=89,         VMSPR2*
                 SYSPGNM=(0-4,14-33,3840-3903),                    VMSPR2*
                 SYSHRSG=(240,241,242,243)                         VMSPR2
```

Note that both saved systems have been expanded by one segment and that the location of the small saved system has been changed.

Modifying the two loadlist EXECs involves the same tedious process we went through to modify SYSTEM NETID. The virgin loadlists must be saved on the CMS base source disk. Then, auxfiles must be created to list the updates for sequencing and modifying the loadlists. Finally, updated copies are built on the A-disk and moved to the S-disk. The one extra twist in this case is that you must also create new SLC files if there don't happen to be existing ones for the addresses you need. (Remember that you need a X'02' immediately preceding the characters "SLC".) The files and commands needed for making these changes to the two CMS nuclei follow:

FILE: CMSLOAD AUXLCL

```
 *$*$*$*$*$*$*$*$*$*$*$*$*$*$*$ AUXLCL $*$*$*$*$*$*$*$*$*$*$*$*$*$*$*$*
 EXPAND - 08/07/82 - EXPAND SMALL NUCLEUS SSTAT/YSTAT BY ONE SEGMENT
 SYSGEN - 08/07/82 - MOVE SMALL NUCLEUS TO FIT ABOVE 5-MEG MACHINE
 SEQUENCE - 08/07/82 - SEQUENCE LOADLIST EXEC SINCE IBM WON'T
```

FILE: CMSLOAD SYSGEN

```
 ./ R 00018000          $ 18100 100
 &1 &2 &3 SLC L530000
 ./ R 00079000          $ 79100 100
 &1 &2 &3 SLC L560000
```

FILE: CMSLOAD EXPAND

```
 ./ R 00079100          $ 79200 100
 &1 &2 &3 SLC L570000
```

FILE: CMSLOADL AUXLCL

```
 *$*$*$*$*$*$*$*$*$*$*$*$*$*$*$*$ AUXLCL $*$*$*$*$*$*$*$*$*$*$*$*$*$*$*$*
 EXPAND - 08/07/82 - EXPAND LARGE NUCLEUS SSTAT/YSTAT BY ONE SEGMENT
 SEQUENCE - 08/07/82 - SEQUENCE LOADLIST EXEC SINCE IBM WON'T
```

FILE: CMSLOADL EXPAND

```
 ./ R 00079000          $ 79100 100
 &1 &2 &3 SLC LF40000
```

FILE: SLC L530000 S1 SLC L570000 S1 SLC LF40000 S1

```
    +----------------+  +----------------+  +----------------+
    |  bSLC  530000  |  |  bSLC  570000  |  |  bSLC  F40000  |  "b" = X'02'
    +----------------+  +----------------+  +----------------+
```

```
 access 190 f/a
 access 495 d
 copy cmsload exec f = = d ( olddate
 copy cmsloadl exec f = = d ( olddate
 access 495 d/a
 update cmsload exec d dmssplcl ( ctl rep
 update cmsloadl exec d dmssplcl ( ctl rep
 access 190 f
 copy cmsload exec a = = f1 ( olddate replace
 copy cmsloadl exec a = = f1 ( olddate replace
 access 190 f/a
```

When you have gotten the loadlists updated, rebuild your CP system with the new definitions in DMKSNT and then rebuild the CMS nuclei and saved systems, using the same commands you used in building them originally. When you are ready to install a new PUT, you will have to apply these same updates to the loadlists from the PUT. Having the auxfiles and the update files sitting on your A-disk should serve as a reminder to do this.

G. Notes on Updating a Production SP2 CMS System

Because lots of new function tends to mean lots of new bugs, you'll want to be sure to get the IBM and VMSHARE buckets for the new system and apply the corrective service suggested there. When you are happy with the new system, you can make it available to your users to try out. Eventually, you'll want to make it the production CMS system. The procedure will be exactly the same as the one used in converting the "NEWCMS" system to "CMS" back on Release 1, except that now you may be putting both "CMS" and "CMSL" into production.

Once it's in production, though, you're likely to encounter a few more glitches, so I'll tell you what I've learned so far about updating an SP2 CMS on the fly. Most of what I told you about updating SP1 CMS still applies, but there are some changes.

The situation with the HELP files is completely different, since they are no longer on the S-disk. Changing the HELP disk does not cause the problems for HELP that changing the S-disk caused in Release 1; that is, your users no longer get I/O error messages because their disk origin pointer is invalid. Anyone who uses HELP has the HELP disk accessed and has a directory of the mode 2 HELP files in his virtual memory. Therefore, if you want to change a mode 2 HELP file, you should rename it, so that it can still be found at its old location by users who have old directories. The mode 1 HELP files, which are the vast majority, are searched for every time the user enters HELP, so you can add or replace them more easily, but it's still better to rename one you are replacing, rather than erasing it, in case someone else is reading it right at that moment. If you add a mode 2 HELP file, users with old HELP disk directories will be able to find it.

The situation with the shared SSTAT and YSTAT has also changed. Since there is no longer a CMSZER shared segment, you no longer do CMSZGENs or DMSZESs to get the shared SSTAT and YSTAT back in synch with the S- and Y-disks. The shared SSTAT and YSTAT are now part of the CMS saved system. So, when you change the S-disk or the Y-disk, you must do SAVESYSs to resave your CMS saved systems. In SP2 your users get the message DMSINS100W SHARED SSTAT/YSTAT NOT AVAILABLE any time you make a change to the S-disk or Y-disk, even if it's only a change to a mode 1 file. This is because DMSINS now checks for whether the shared SSTAT and YSTAT are current by comparing the date the disk was last updated with the disk update date saved in the saved system. This makes changing the system disks on the fly less graceful than it used to be. You will even find that you make the shared SSTAT or YSTAT unavailable by simply accessing the S- or Y-disk in R/W mode and releasing it, without making any changes.

One other difference which affects the process of updating a production CMS system is that CMSSEG is no longer detached after every use. As far as I have been able to learn, the other rules for changing CMS still apply.

H. Miscellaneous Caveats

There are a few other points you should be aware of:

The base source for Release 2 corresponds to SLU 12 for Release 1. If you are at a higher level than that on SP1, you may find that you need to go back and apply some of the same corrective service to SP2 that you had applied to SP1 when you were at an earlier level.

A few of the SP2 CMS base source files were sequenced improperly; there are fixes for this problem on the first SP2 PUT.

In SP2, a number of CMS commands are implemented as nucleus extensions. If you are testing a fix or mod to one of these, you may have to do a NUCXDROP to get rid of the old version before you can test your new version. When in doubt, do a NUCXMAP.

If you have a large number of mode 2 files on your S- and Y-disks, you may find that you cannot IPL an SP2 CMS system by address. The symptom is messages DMSFRE159T and DMSFRE162T. The circumvention is to DETACH your 19E disk. The fix is VM16392, which has a pre-requisite of VM16189.

Some other APARs which may be of interest include: VM16450, which is a performance fix for FILELIST, and VM16189, which is a fix for abends caused by IPLing an SP2 CMS system by name and specifying an IPL parm greater than thirty-six characters in length.

If you use the new SENDFILE command to send data to an SP1 CMS system, be sure to specify the "OLD" parameter in the SENDFILE command.

If your local mods or programs stop working, two things to check out are the fact that the handling of mode bytes is different in SP2 and the fact that the "CONCCWS" field has been moved from NUCON to OPSECT. In general, programs which reference fields in OPSECT need to be reassembled.

You may also have trouble with a couple of widely-used mods from the Waterloo tape. In particular, RESLIB doesn't seem to work unless the CMS nucleus is at the top of the virtual machine or beyond it. The Cornell CARD and FCOPY modules won't run correctly on an SP2 CMS system unless you put a dummy DMSZES MODULE on your S-disk.

APPENDIX A

Applying Service to Other Levels of VM

Naming Conventions for Current Levels of VM

	REL 6	BSEPP 2	SEPP 2	SP 1	HPO 1	SP 2
TEXT FILES	TEXT TXTAP	TEXT TXTAP	TEXT TXTAP	TEXT TXTAP TXTMP	TXTH1 TXTH1A TXTHIM	TEXT TXTAP TXTMP
AUXFILES	AUXR60 AUXM60	AUXB20 AUXMB20	AUXS20	AUXSP AUXSP11 AUXMSP AUXMSP11	AUXHP1	AUXSP AUXMSP
UPDATES	RnnnnnDx MnnnnnDS	NnnnnnDx CnnnnnDS	KnnnnnDK	SnnnnnDx EnnnnnDS	AnnnnnDK	LnnnnnDx ZnnnnnDS
CP MACLIBS	DMKMAC DMKAMAC	CPBSE DMKMAC DMKAMAC	CPSEP DMKMAC DMKAMAC	DMKSP DMKSPA DMKSPM DMKMAC	DMKH1 DMKSP DMKSPA DMKSPM DMKMAC	DMKSP DMKSPA DMKSPM DMKMAC
CMS MACLIBS	CMSLIB	CMSBSE CMSLIB		DMSSP CMSLIB		DMSSP CMSLIB
CP CNTRL FILES	DMKR60 DMKR6A	DMKB20 DMKB2A	DMKS20 DMKS2A	DMKSP DMKSPA DMKSPM	DMKH1 DMKH1A DMKH1M	DMKSP DMKSPA DMKSPM
CMS CNTRL FILES	DMSR60 DMSM60	DMSB20 DMSMB20		DMSSP DMSMSP		DMSSP DMSMSP

Service Disk Layouts for "Delta" Systems

Some VM systems are shipped to you as "deltas" to another system. Specifically, BSEPP Release 2 and SEPP Release 2 are shipped in the form of updates to be applied on top of a VM Release 6 system, and HPO Release 1 is shipped as updates to be applied on top of a VM/SP Release 1 system. This makes applying service slightly more complicated for these systems than it is for VM/SP. Note that neither SEPP nor HPO Release 1 "versions" CMS, so you use the next lower system level for CMS, i.e., you use VM/SP CMS with HPO and BSEPP CMS with SEPP.

The standard way of maintaining a delta system is simply to load the corresponding files for the two pieces of the system onto the same service disks. Thus, in the case of an HPO system, the base disk should include the base ASSEMBLE, COPY, MACRO, TEXT, TXTAP, TXTMP, UPDTAP, and UPDTMP files from the base VM/SP system. It should also contain the original UPDTHP1 files from the HPO distribution tape, as well as the textfiles from that tape (TXTH1 and TXTH1A). The PUT disk should contain the SP auxfiles, updates, and textfiles from the SP portion of the PUT, plus the HPO auxfiles, updates, and textfiles from the HPO portion of the PUT. The HPO auxfiles are complete replacements for the SP auxfiles, but only for the "HPO-versioned" modules. An HPO auxfile may list both SP fixes (S12345DK) and HPO fixes (A12222DK). In some cases, there will be two different versions of the same fix for SP and HPO; these may be distinguished by the prefix on the filetypes.

Exactly the same scheme is used with a BSEPP or SEPP system. You load the Release 6 tape and the base (B)SEPP tape onto your base service disk, and you load the Release 6 files and the (B)SEPP files from the PUT onto your PUT disk. There is one difference, however, from the HPO scheme; all of IBM's textfiles for a (B)SEPP system are named TEXT or TXTAP. Originally, the (B)SEPP-versioned textfiles were named TXTB20 and TXTB2A or TXTS20 and TXTS2A. But then somebody at IBM who didn't understand what was going on decided that the (B)SEPP-versioned textfiles should be renamed. This blunder has been causing a good many systems around the world to crash in obscure ways for a couple of years now. If you are maintaining a (B)SEPP system, be very careful to load the Release 6 textfiles from a given PUT before you load the (B)SEPP textfiles, so that the (B)SEPP textfiles will overlay the corresponding vanilla textfiles, rather than the other way around. And don't use the SELECT option of VMFPLC2 LOAD, either. (That's the option that says not to replace a file on the disk with a file from the tape with the same name if they have the same timestamp.) There have been cases of the vanilla and (B)SEPP textfiles having the same timestamp; if SELECT is specified in such a case, the resulting system is missing a (B)SEPP update and will tend to be unstable. To avoid such problems, many (B)SEPP installations have taken to loading the (B)SEPP textfiles from the PUT onto one minidisk and the vanilla textfiles from the PUT onto another disk later in their standard disk search order.

Preferred Auxfiles

One new concept you need to master in order to understand how to apply service to a delta system is the "preferred auxfile". As mentioned above, the delta systems provide auxfiles which completely replace the corresponding base system auxfiles, but only for those modules which are "versioned" by the delta system. This means that there needs to be a way of saying in the control file that the base auxfile should be used only if there is no delta auxfile for that module. This is done by specifying a preferred auxfile, as in the CP control file for BSEPP Release 2:

FILE: DMKB20 CNTRL

```
 TEXT MACS ....
 TEXT AUXB20
 TEXT AUXR60 AUXB20
```

That last line means that the updates listed in the AUXR60 auxfile are to be applied unless there is an AUXB20 auxfile for the module, in which case the whole line is to be ignored. The second line from the bottom then says to apply the updates listed in the AUXB20 auxfile, if there is one. There can be more than one preferred auxfile listed in a control file statement. If any of the preferred auxfiles exists for the module, then the statement is skipped.

One further wrinkle in using UPDATE with control files: UPDATE records each auxfile name as it is being used and does not use it again, even if the control file specifies that auxfile again. This can be seen in the following control file, which is used for HPO Release 1 for MP systems. It is included as an exercise for the reader:

FILE: DMKH1M CNTRL

```
 TEXT MACS ....
 H1M  AUXH1
 H1A  AUXH1 UPDTMP
 H1   AUXH1 UPDTAP UPDTMP
 MP   AUXSP11 AUXH1
 AP   AUXSP11 AUXH1 UPDTMP
 TEXT AUXSP11 AUXH1 UPDTAP UPDTMP
 MP   UPDTMP
 AP   UPDTAP
 TEXT AUXSP
```

Hint: All HPO updates are listed in auxfiles named AUXH1, but it must be possible to use this control file to produce HPO-versioned textfiles named TXTH1, TXTH1A, or TXTH1M (as appropriate) and to produce textfiles for non-versioned modules named TEXT, TXTAP, or TXTMP (as appropriate).

APPENDIX B

The FRE013 Trap

This version of the "FRE013 trap" is current for a VM/SP Release 1, SLU 7, system running on a UP without ECPS microcode. Versions of this trap for other configurations are available from the IBM Support Center.

FILE: DMKFRE FRE013

```
./ I 1680000 $ 1680100 100
         EJECT
************DMKFRE FRE013 TRAP FOR FREE STORAGE PROBLEMS**************
*                      VM/SP VERSION - UNI-PROCESSOR                  *
*********************************************************************
*****    MUST ALSO TURN OFF DMKDSP1, DMKDSP2, DMKUNTFR ECPS       ******
*****     INSTRUCTIONS IN DMKDSP AND DMKUNT FOR MICROCODED        ******
*****    CPU TYPES 3138, 3148, 303X, 4331, 4341, ETC.             ******
*********************************************************************
*****    THIS VERSION HAS THE FOLLOWING PRE-REQUISITES IN ORDER   ******
*****    TO INSTALL AND EXECUTE SUCCESSFULLY:                     ******
*****    VM13017 PUTXXXX UVXXXXX - ADD LARGER BUFFER TO SUBPOOL   ******
*****    VM14280 PUTXXXX UVXXXXX - MICROCODE FOR TCH/FREE/FRET    ******
*****    VM12640 PUTXXXX UVXXXXX - CLEAN UP EXTENDED PAGES        ******
*********************************************************************
* TRAP OPERATION:                                                   *
* A) TURNS OFF ECPS FOR 'FREE' AND 'FRET'                           *
* B) FOR EACH 'FREE' REQUEST:                                       *
*    1) EXPANDS EACH STORAGE REQUEST BY TWO DOUBLEWORDS             *
*    2) PUTS 'GVN' + REQUEST LENGTH + CALLER'S R14 IN THE FIRST     *
*       ADDITIONAL DOUBLEWORD                                       *
*    3) IF THE CALLER IS PAGEABLE, GETS THE MODULE NAME AT THE START *
*       OF THE PAGE IN WHICH THE MODULE RESIDES, AND PUTS IT IN THE  *
*       SECOND ADDITIONAL DOUBLEWORD                                *
*    4) RESETS FREE STORAGE BLOCK TO X'EEEEEE...'                   *
* C) FOR EACH 'FRET' REQUEST:                                       *
*    1) CHECKS THE REQUEST LENGTH AGAINST THE SAVED LENGTH          *
*    2) CHECKS TO SEE THAT THE EYECATCHER IS 'GVN' IN THE FIRST     *
*       ADDITIONAL DOUBLEWORD                                       *
* D) ABENDS FROM ILLEGAL SITUATIONS (FRET REQUESTS ONLY):           *
*    ABENDFRE013 - STORAGE NOT MARKED 'GVN' ON CALL TO FRET         *
*    ABENDFRE033 - FRET SIZE NOT THE SAME AS EYECATCHER SIZE        *
*********************************************************************
./ R 2965000 $ 2966000
         DC    X'070007000700'  SHUT OFF ECPS 'FREE'          @FRE13SP
./ I 3090000 $ 3091000
         LA    R2,2(,R2)      ADD 2 TO REQUESTED SIZE         @FRE13SP
./ I 3280000 $ 3281000 1000
         L     R15,SUBSIZES(R7)  SET TO FULL SUBPOOL SIZE     @FRE13SP
         SLL   R15,3          TIMES 8 FOR NUMBER OF BYTES     @FRE13SP
         B     FREE20CN                                        @FRE13SP
```

```
./ I 3300000 $ 3300100 100
**********************  FRE013 TRAP  ************************ @FRE13SP
*** FILL BLOCK WITH X'EEEEEEEE' S                             @FRE13SP
*** MOVE EYECATCHER INTO FIRST WORD AFTER REQUESTED STORAGE,   @FRE13SP
***      CALLER'S R14 + LENGTH (FROM R0) INTO THE 2ND WORD     @FRE13SP
***      CALLING PGM'S NAME (IF PGM IS PAGEABLE) INTO WORDS 3+4 @FRE13SP
          L     R15,FREEWORK   GET FULL SUBPOOL OR BLOCK SIZE   @FRE13SP
FREE20CN  L     R14,GPR1       POINT TO FREE STORAGE            @FRE13SP
          L     R2,FREER0      GET ORIGINAL STORAGE REQUEST     @FRE13SP
          SLL   R2,3           GET NUMBER OF REQUESTED BYTES    @FRE13SP
          AR    R2,R14         POINT TO END OF REQSTD STORAGE   @FRE13SP
          L     R1,TESTPATT    SET UP PATTERN OF EEEEEEEE'S     @FRE13SP
          MVCL  R14,R0         CLEAR THE STORAGE TO EEEEEE'S    @FRE13SP
          L     R3,EYEGVN      AND SET UP TRAP DATA             @FRE13SP
          L     R4,FREER14     POINT TO CALLER'S RETN REGISTER  @FRE13SP
          CL    R4,APAGCP      IS CALLER PAGEABLE?              @FRE13SP
          BL    NOTPAG         NO, SKIP SAVING THE MODULE NAME  @FRE13SP
          N     R4,XPAGNUM     YES, FIND BEGINNING OF MODULE    @FRE13SP
          LM    R6,R7,0(R4)    GET MODULE NAME AT BEG OF MODULE @FRE13SP
          STM   R6,R7,8(R2)    SAVE IT BEHIND THE EYECATCHER    @FRE13SP
          L     R4,FREER14     RESET R4 TO CALLER'S R14         @FRE13SP
NOTPAG    DS    0H             HERE IF CALLER IS NOT PAGEABLE   @FRE13SP
          ICM   R4,8,FREER0+3  INCLUDE COUNT IN DOUBLE WORDS    @FRE13SP
          STM   R3,R4,0(R2)    SAVE CONSTANT AND RETURN REG     @FRE13SP
./ I 5130000 $ 5131000
          ST    R2,FREEWORK   SAVE FULL BYTE CNT OF STORAGE BLK @FRE13SP
./ I 6110000 $ 6111000
          ST    R2,FREEWORK   SAVE FULL BYTE CNT OF STORAGE BLK @FRE13SP
./ I 8610000 $ 8611000
          ST    R2,FREEWORK   SAVE FULL BYTE CNT OF STORAGE BLK @FRE13SP
./ I 11462500 $ 11462600
          LA    R10,FR14       SET R10 TO GO TO FRE013 TRAP     @FRE13SP
./ R 11525000 $ 11526000
          DC    X'070007000700'  SHUT OFF ECPS 'FRET'           @FRE13SP
./ R 11550000 $ 11551000 1000
          LM    R7,R9,ADCONFRT  INIT R7-R9 FOR FRET PROCESS     @FRE13SP
          LA    R10,FR14       SET R10 TO GO TO FRE013 TRAP     @FRE13SP
./ I 11950000 $ 11950100 100
FR14      DS    0H             START OF FRE13 TRAP CATCH LOGIC  @FRE13SP
          LR    R7,R0          SIZE INTO REG7                   @FRE13SP
          SLL   R7,3           CONVERT TO BYTES                 @FRE13SP
          L     R8,EYEGVN      GET THE 'GVN' EYECATCHER         @FRE13SP
          C     R8,0(R7,R1)    DID WE FIND 'GVN'?               @FRE13SP
          BE    CHECKSZ        YES, CONTINUE                    @FRE13SP
          ABEND 13             NO, ABENDFRE013                  @FRE13SP
CHECKSZ   DS    0H             CHECK THE SIZE                   @FRE13SP
          SR    R8,R8          ZERO REG8                        @FRE13SP
          IC    R8,4(R7,R1)    GET SIZE FROM EYECATCHER         @FRE13SP
          LR    R9,R0          GET SIZE PASSED TO DMKFRET       @FRE13SP
          N     R9,SIZEMASK    STRIP OFF ALL BUT LAST BYTE      @FRE13SP
          CLR   R9,R8          IS IT THE SAME?                  @FRE13SP
          BE    SETEYE         YES - GO RESET EYECATCHER        @FRE13SP
          ABEND 33             NO - ABEND FRE033                @FRE13SP
SETEYE    DS    0H             RESET EYECATCHER                 @FRE13SP
```

```
            L     R8,EYEFREE     REPLACE 'GVN' WITH 'FREE'        @FRE13SP
            ST    R8,0(R7,R1)    TO TRAP 2ND FRET OF SAME BLOCK   @FRE13SP
            AL    R0,F2          BUMP COUNT UP TO FULL FRET SIZE  @FRE13SP
            LR    R2,R0          FRET01 NEEDS LENGTH IN REG2
            LA    R9,DMKFRETE    GET ADDRESS OF FRETE
            C     R9,FREER15     DID WE COME IN AT FRETE?
            BE    FRET01         YES, GO TO FRET01 TO FINISH
            LM    R7,R10,ADCONFRT RELOAD REGS FOR FRET PROCESS    @FRE13SP
            BR    R10            GO COMPLETE FRET PROCESSING      @FRE13SP
            EJECT
./ I 15010000 $ 15011000 1000
********** CONSTANTS FOR FRE13 TRAP **************************
            DS    0F             ...FOR ALIGNMENT                 @FRE13SP
SIZEMASK    DC    XL4'000000FF'  TO ISOLATE LAST BYTE OF SIZE     @FRE13SP
TESTPATT    DC    XL4'EE000000'  PATTERN CHARACTERS FOR GIVEN AREA@FRE13SP
EYEGVN      DC    XL4'9AC7E5D5'  EYECATCHER 'GVN'                 @FRE13SP
EYEFREE     DC    XL4'C6D9C5C5'  EYECATCHER 'FREE'                @FRE13SP
```

FILE: DMKFRE FRE13X

```
./ * MODIFY FRE013 TRAP TO ADD ONLY ONE DOUBLEWORD, RATHER THAN  FRE13X
./ * TWO,  TO EACH FREE REQUEST.                                 FRE13X
./ *                                                             FRE13X
./  R  03091000 $ 03091100 100                                   FRE13X
            LA    R2,1(,R2)          ADD 1 TO REQUESTED SIZE.    FRE13X
./  D  03300500 $                                                FRE13X
./  D  03301500 03302100 $                                       FRE13X
./  R  11951900 $ 11951910 10                                    FRE13X
            AL    R0,F1              BUMP COUNT TO FULL FRET SIZE. FRE13X
```

FILE: DMKFRE FRE13Y

```
./ * MODIFY FRE013 TRAP NOT TO SET STORAGE TO X'EE'.             FRE13Y
./ *                                                             FRE13Y
./  D  03281000 03282000 $                                       FRE13Y
./  D  03300200 $                                                FRE13Y
./  D  03300600 $                                                FRE13Y
./  D  03301100 03301200 $                                       FRE13Y
./  D  15014000 $                                                FRE13Y
```

APPENDIX C

Mod to Resolve External References in Small CP Nucleus

This modification is for VM/SP Release 1, SLU 7. It is based on a suggestion from Jim Best (PWC). It builds a dummy module, DMKRES, which contains ENTRY statements for all the external references which are normally unresolved in a CP nucleus built with the "small CP option". This zero-length module must be placed at address zero, so that these references are "resolved" to zeroes. WARNING: If you put DMKRES at any address other than zero, your system will crash a lot.

FILE: CPLOADSM SMALCP

```
 ./ * INSERT THE DUMMY MODULE 'DMKRES' BETWEEN DMKLD00E AND       SMALCP
 ./ * DMKPSA IN THE SMALL CP LOADLIST, IN ORDER TO RESOLVE        SMALCP
 ./ * ALL THE NORMALLY-UNRESOLVED EXTERNAL REFERENCES IN THE      SMALCP
 ./ * SMALL CP NUCLEUS TO ZERO.                                   SMALCP
 ./ *                                                             SMALCP
 ./  I  00006000 $ 00006500 500                                   SMALCP
 &1 &2 &3 DMKRES
```

FILE: DMKRES ASSEMBLE

```
 RES      TITLE 'DMKRES - RESOLVE NORMALLY UNRESOLVED REFERENCES' SMALCP
 DMKRES   CSECT ,                                                 SMALCP
          SPACE 1                                                 SMALCP
          ENTRY DMKBSCER,DMKDADER,DMKFPS                          SMALCP
          ENTRY DMKMHCIN,DMKMHCRE,DMKMHVSM                        SMALCP
          ENTRY DMKQVMCU,DMKQVMEP,DMKQVMRT,DMKQVMTS               SMALCP
          ENTRY DMKRGAIN,DMKRGBEN,DMKRGBFM,DMKRGBIC               SMALCP
          ENTRY DMKRGC,DMKRGDOB,DMKRGDOI                          SMALCP
          ENTRY DMKRNH,DMKRNHIC,DMKRNHIN,DMKRNHND                 SMALCP
          ENTRY DMKRNHTR,DMKSLC,DMKSNTRN                          SMALCP
          ENTRY DMKSNTQN                                          SMALCP
          SPACE 1                                                 SMALCP
          ENTRY DMKSSS,DMKSSSHV,DMKSSSMQ,DMKSSSEN,DMKSSSVA        SMALCP
          ENTRY DMKSSSDE,DMKSSSL1,DMKSSSL2,DMKSSSL3               SMALCP
          ENTRY DMKSSSUS,DMKSSSAS,DMKSSSLN,DMKSSSRL               SMALCP
          ENTRY DMKSSSVM,DMKSSTHV,DMKSSTUS                        SMALCP
          ENTRY DMKSSUI1,DMKSSUI2,DMKSSUCF,DMKSSULO               SMALCP
          SPACE 1                                                 SMALCP
          ENTRY DMKTRKIN,DMKTRKFP,DMKTRKVA                        SMALCP
          SPACE 1                                                 SMALCP
          ENTRY DMKVCPIL,DMKVCRMT,DMKVCRNR,DMKVCRRD,DMKVCRWT      SMALCP
          ENTRY DMKVCT,DMKVCTCH,DMKVCTCN,DMKVCTDA,DMKVCTEN        SMALCP
          ENTRY DMKVCTER,DMKVCTLO,DMKVCTQS                        SMALCP
          ENTRY DMKVCTRM,DMKVCTSV,DMKVCVER,DMKVCXIO               SMALCP
```

```
          ENTRY DMKVSC,DMKVSCVR                                  SMALCP
*                                                                SMALCP
DMKBSCER  EQU   *                                                SMALCP
DMKDADER  EQU   *                                                SMALCP
DMKFPS    EQU   *                                                SMALCP
DMKMHCIN  EQU   *                                                SMALCP
DMKMHCRE  EQU   *                                                SMALCP
DMKMHVSM  EQU   *                                                SMALCP
DMKQVMCU  EQU   *                                                SMALCP
DMKQVMEP  EQU   *                                                SMALCP
DMKQVMRT  EQU   *                                                SMALCP
DMKQVMTS  EQU   *                                                SMALCP
DMKRGAIN  EQU   *                                                SMALCP
DMKRGBEN  EQU   *                                                SMALCP
DMKRGBFM  EQU   *                                                SMALCP
DMKRGBIC  EQU   *                                                SMALCP
DMKRGC    EQU   *                                                SMALCP
DMKRGDOB  EQU   *                                                SMALCP
DMKRGDOI  EQU   *                                                SMALCP
DMKRNH    EQU   *                                                SMALCP
DMKRNHIC  EQU   *                                                SMALCP
DMKRNHIN  EQU   *                                                SMALCP
DMKRNHND  EQU   *                                                SMALCP
DMKRNHTR  EQU   *                                                SMALCP
DMKSLC    EQU   *                                                SMALCP
DMKSNTRN  EQU   *                                                SMALCP
DMKSNTQN  EQU   *                                                SMALCP
          SPACE 1                                                SMALCP
DMKSSS    EQU   *                                                SMALCP
DMKSSSAS  EQU   *                                                SMALCP
DMKSSSDE  EQU   *                                                SMALCP
DMKSSSEN  EQU   *                                                SMALCP
DMKSSSHV  EQU   *                                                SMALCP
DMKSSSLN  EQU   *                                                SMALCP
DMKSSSL1  EQU   *                                                SMALCP
DMKSSSL2  EQU   *                                                SMALCP
DMKSSSL3  EQU   *                                                SMALCP
DMKSSSMQ  EQU   *                                                SMALCP
DMKSSSRL  EQU   *                                                SMALCP
DMKSSSUS  EQU   *                                                SMALCP
DMKSSSVA  EQU   *                                                SMALCP
DMKSSSVM  EQU   *                                                SMALCP
DMKSSTHV  EQU   *                                                SMALCP
DMKSSTUS  EQU   *                                                SMALCP
DMKSSUCF  EQU   *                                                SMALCP
DMKSSUI1  EQU   *                                                SMALCP
DMKSSUI2  EQU   *                                                SMALCP
DMKSSULO  EQU   *                                                SMALCP
          SPACE 1                                                SMALCP
DMKTRKIN  EQU   *                                                SMALCP
DMKTRKFP  EQU   *                                                SMALCP
DMKTRKVA  EQU   *                                                SMALCP
          SPACE 1                                                SMALCP
DMKVCPIL  EQU   *                                                SMALCP
```

```
DMKVCRMT EQU   *                                                SMALCP
DMKVCRNR EQU   *                                                SMALCP
DMKVCRRD EQU   *                                                SMALCP
DMKVCRWT EQU   *                                                SMALCP
DMKVCT   EQU   *                                                SMALCP
DMKVCTCH EQU   *                                                SMALCP
DMKVCTCN EQU   *                                                SMALCP
DMKVCTDA EQU   *                                                SMALCP
DMKVCTEN EQU   *                                                SMALCP
DMKVCTER EQU   *                                                SMALCP
DMKVCTLO EQU   *                                                SMALCP
DMKVCTQS EQU   *                                                SMALCP
DMKVCTRM EQU   *                                                SMALCP
DMKVCTSV EQU   *                                                SMALCP
DMKVCVER EQU   *                                                SMALCP
DMKVCXIO EQU   *                                                SMALCP
DMKVSC   EQU   *                                                SMALCP
DMKVSCVR EQU   *                                                SMALCP
         SPACE 1                                                SMALCP
         END   DMKRES                                           SMALCP
```

FILE: DTVTAB SMALCP

```
 ./ * MAKE IPCS/E UNDERSTAND THAT DMKRES IS FIRST CP MODULE,    SMALCP
 ./ * RATHER THAN DMKPSA.                                       SMALCP
 ./ *                                                           SMALCP
 ./ R 00090000          $ 90490 490                             SMALCP
 CPFIRST  DC    CL8'DMKRES'         1ST CP NUC MAP MODULE NAME. SMALCP
```

APPENDIX D

Alternate Nucleus Mod, System Numbering Mod, and EXECs for Building and Installing CP

Please note that these modifications and EXECs are included as an example. No warranty is implied. These modifications are from a VM/SP Release 1, SLU 11, system and have been used for both FBA and CKD sysres devices, on MP and UP systems.

It should be noted that all "alternate nuclei" must be on CP-owned volumes. All nuclei must have the same cylinder/block location, the one specified in the SYSNUC parm on the SYSRES macro in DMKSYS ASSEMBLE.

FILE: DMKCKP ALTNC0 A1

```
 ./ * PURPOSE:                                                    ALTNC0
 ./ *  TO SUPPORT THE 'ALTERNATE NUCLEUS MODIFICATION'.  SPECI-   ALTNC0
 ./ *  FICALLY, TO DIFFERENTIATE BETWEEN THE SYSRES ADDRESS AND   ALTNC0
 ./ *  THE SYSIPL ADDRESS WHEN CHECKPOINTING AND WARM STARTING,   ALTNC0
 ./ *  RESPECTIVELY.                                             ALTNC0
 ./ *                                                            ALTNC0
 ./ R 00280000           $ 00280010  00000010                    ALTNC0
 *        SYSWARM AREA OF THE SYSRES PACK.                       ALTNC0
 ./ R  02790000  02800000 $  02790010  00000010                  ALTNC0
          LH    R0,SYSIPLDV    GET SYSRES ADDRESS                ALTNC0
          ST    R0,SYSRES      SAVE FOR USE BY IBM CODE          ALTNC0
          LH    R0,INTTIO      GET IPL ADDR LEFT BY 'DMP' OR IPL ALTNC0
          STH   R0,SYSIPL      AND SAVE FOR ALT. NUCLEUS CODE.   ALTNC0
 ./ R  05080000           $  05080010                            ALTNC0
          LH    R2,SYSIPL      GET IPL DEVICE ADDRESS            ALTNC0
 ./ R  09020000           $  09020010  00000010                  ALTNC0
 SYSRES   DS    F              SYSTEM RESIDENCE ADDRESS          ALTNC0
 SYSIPL   DS    F              SYSTEM IPL ADDRESS                ALTNC0
 *                                                               ALTNC0
```

FILE: DMKCPI ALTNC0 A1

```
 ./ * PURPOSE:                                                    ALTNC0
 ./ *  TO SUPPORT THE 'ALTERNATE NUCLEUS MODIFICATION'.  SPECI-   ALTNC0
 ./ *  FICALLY, TO ELIMINATE THE TEST FOR IPL'ING FROM THE        ALTNC0
 ./ *  'SYSRES' VOLUME AND TO SAVE THE ACTUAL IPL ADDRESS IN      ALTNC0
 ./ *  'PSAIPLDV' FOR LATER USE BY DMKCKP, DMKCPS AND DMKDMP.     ALTNC0
 ./ *                                                            ALTNC0
 ./ R 05950000 05970000 $ 5959990 9990                           ALTNC0
          STH   R10,PSAIPLDV   SAVE IPL ADDRESS.                 ALTNC0
 ./ R 06450000           $ 6454990 4990                          ALTNC0
          LH    R15,PSAIPLDV   GET ADDRESS OF IPL DEVICE.        ALTNC0
```

```
./ R 09990000          $ 9994990 4990                              ALTNC0
         LH    R1,PSAIPLDV    ADDRESS OF IPL DEVICE...            ALTNC0
./ R 10970000          $ 10974990 4990                             ALTNC0
         CH    R15,PSAIPLDV   IPL DEVICE?                         ALTNC0
./ R 18100000 18110000 $ 18100990 990                              ALTNC0
         LR    R8,R1          STANDARD ADDR FOR RDEVBLOK.         ALTNC0
         CALL  DMKSCNRD       GET SYSRES DEVICE ADDRESS.          ALTNC0
         STH   R1,SYSIPLDV    SAVE A(CKPT/WARM/ERROR VOLUME).     ALTNC0
         TM    APSTAT1,APUOPER  OTHER PROCESSOR OPERATIONAL?      ALTNC0
         BZ    CPIPU020       NO, THEN NO PREFIXING.              ALTNC0
         L     R15,PREFIXA    YES, GET ABSOLUTE PSA.              ALTNC0
         STH   R1,SYSIPLDV-PSA(,R15)  SAVE VOLUME ADDRESS THERE.  ALTNC0
         L     R15,PREFIXB    AND GET OTHER PROC'S PSA, TOO.      ALTNC0
         STH   R1,SYSIPLDV-PSA(,R15) SAVE VOLUME ADDRESS FOR HIM.ALTNC0
CPIPU020 L     R15,CPIDMPSD   ALSO SAVE A(CKPT/WARM/ERROR...      ALTNC0
         STH   R1,0(,R15)     ...VOLUME) IN 'DMKDMPSD'.           ALTNC0
         LH    R1,PSAIPLDV    LOAD ADDRESS OF IPL DEVICE.         ALTNC0
./ D 18140000 18150000                                             ALTNC0
```

FILE: DMKCPS ALTNC0 A1

```
./ * PURPOSE:                                                      ALTNC0
./ *  TO SUPPORT THE 'ALTERNATE NUCLEUS MODIFICATION'.  SPECI-    ALTNC0
./ *  FICALLY, TO ALLOW FOR SAVING THE PATH TO AN IPL VOLUME      ALTNC0
./ *  WHICH IS NOT THE ONE DESIGNATED AS THE 'SYSRES' VOLUME.     ALTNC0
./ *                                                              ALTNC0
./  R  05190000        $ 05194990 4990                             ALTNC0
CPSDMPSD DC    A(PSAIPLDV-PSA)  PNTR TO CCU ADDR OF SYS IPL VOL. ALTNC0
```

FILE: DMKDDR ALTNC0 A1

```
./ * PURPOSE:                                                      ALTNC0
./ *  TO SUPPORT THE 'ALTERNATE NUCLEUS MODIFICATION'.  SPECI-    ALTNC0
./ *  FICALLY, TO ALLOW FOR COPYING A NUCLEUS TO A VOLUME WITH    ALTNC0
./ *  A DIFFERENT VOLID.  PREVIOUSLY, THIS RESULTED IN MESSAGE:   ALTNC0
./ *  DMKDDR722E OUTPUT UNIT NOT PROPERLY FORMATTED FOR NUCLEUS.  ALTNC0
./ *  NOTE!!!  THE LOCATION OF THE NUCLEUS CYLINDERS IS STILL     ALTNC0
./ *  THAT DEFINED BY THE SYSRES MACRO; ONLY THE VOLID IS         ALTNC0
./ *  IGNORED, AND THE VOLUME MUST STILL BE CP-OWNED.             ALTNC0
./ *                                                              ALTNC0
./ I 22700000           $ 22700100  00000100                       ALTNC0
*                 NOTE THAT THIS CHECK HAS BEEN MODIFIED         ALTNC0
*                 TO ALLOW THE VOLID TO BE DIFFERENT FROM        ALTNC0
*                 FROM THAT ON THE INPUT NUCLEUS.                ALTNC0
./ D  23410000  23430000 $                                        ALTNC0
```

FILE: DMKDMP ALTNC0 A1

```
./ * PURPOSE:                                                    ALTNC0
./ *  TO SUPPORT THE 'ALTERNATE NUCLEUS MODIFICATION'.  SPECI-   ALTNC0
./ *  FICALLY, TO DIFFERENTIATE BETWEEN THE SYSRES ADDRESS AND   ALTNC0
./ *  THE SYSIPL ADDRESS WHEN CHECKPOINTING AND WARM STARTING,   ALTNC0
./ *  RESPECTIVELY.                                             ALTNC0
./ *                                                           ALTNC0
./ R 08950000          $ 8954990 4990                          ALTNC0
         LH    R1,DMKDMPSD    A(CKPT/WARM/ERROR VOLUME).       ALTNC0
./ R 09460000          $ 9464990 4990                          ALTNC0
         LH    R1,DMKDMPSD    A(CKPT/WARM/ERROR VOLUME).       ALTNC0
./ R 09730000          $ 9734990 4990                          ALTNC0
         LH    R15,DMKDMPSD   A(CKPT/WARM/ERROR VOLUME).       ALTNC0
./ R 12350000          $ 12354990 4990                         ALTNC0
         LH    R1,PSAIPLDV    GET A(SYSTEM IPL DEVICE).        ALTNC0
./ R 12550000          $ 12554990 4990                         ALTNC0
         STH   R1,PSAIPLDV    SAVE PATH'S CCU ADDRESS.         ALTNC0
./ R 12870000          $ 12874990 4990                         ALTNC0
         LH    R15,PSAIPLDV   GET SYSTEM IPL DEVICE ADDRESS.   ALTNC0
./ I 13500000          $ 13505000 5000                         ALTNC0
         STH   R15,DMKDMPSD        WANT THIS ADDRESS IN MSG.   ALTNC0
```

FILE: DMKSAV ALTNC0 A1

```
./ * PURPOSE:                                                    ALTNC0
./ *  TO SUPPORT THE 'ALTERNATE NUCLEUS MODIFICATION'.  SPECI-   ALTNC0
./ *  FICALLY, TO ALLOW FOR WRITING NEW NUCLEI ON VOLUMES NOT    ALTNC0
./ *  DESIGNATED AS THE 'SYSRES' VOLUME.  PREVIOUSLY, THIS      ALTNC0
./ *  RESULTED IN MSG DMKSAV350W DASD 'RADDR' VOLID NOT 'LABEL'. ALTNC0
./ *                                                           ALTNC0
./ *     THIS MOD DELETES BOTH THE LABEL COMPARISON TEST AND    ALTNC0
./ *     THE MESSAGE DMKSAV350W.                               ALTNC0
./ *                                                           ALTNC0
./ R  00460000           $  00460010  00000010                 ALTNC0
*        DMKSAVRS:                                             ALTNC0
*              R10 = IPL DEVICE ADDRESS IN LOW ORDER BYTES     ALTNC0
./  R  01020000 01100000 $ 01020100  00000100                  ALTNC0
*        3. READ THE VOLUME LABEL, BUT DO NOT CHECK FOR        ALTNC0
*           EQUALITY WITH THE SYSRES VOLUME LABEL.             ALTNC0
./  D  06640000 06660000 $                                     ALTNC0
./  D  09640000 09730000 $                                     ALTNC0
```

FILE: PSA ALTNC0 A1

```
./ R 03080000          $ 3081000 1000                          ALTNC0
*        SYSIPLDV IS MAINTAINED IN ALL THREE PSA'S.            ALTNC0
SYSIPLDV DS    1H -      P*3  ADDRESS OF CKPT/WARM/ERROR DEVICE. ALTNC0
./ R 03770000           $ 03770100  00000100                   ALTNC0
*        PSAIPLDV IS MAINTAINED IN ALL THREE PSA'S.            ALTNC0
PSAIPLDV DC    H'0'               SYSTEM IPL DEVICE ADDRESS.   ALTNC0
         DC    H'0'               UNUSED.                      ALTNC0
```

```
FILE: DMKCPE SYSNM0 A1

 ./ I 00220000 $ 00220100 100                                    SYSNM0
         ENTRY DMKCPE#             PU CP SYSTEM #.               SYSNM0
 ./ I 00270000 $ 00270100 100                                    SYSNM0
 DMKCPE#  DC    CL8' 192 '         PU CP SYSTEM NUMBER.          SYSNM0


FILE: DMKCPI SYSNM0 A1

 ./ I  28400000           $  28400100  00000100
         DC    C'; SYSTEM #'                                     SYSNM0
 CPINUM   DC    CL3' '             PU CP SYSTEM NUMBER.          SYSNM0
 ./ I  29850000           $  29850100  00000100
         L     R1,PSACPE#         R1=A(PU CP SYSTEM NUMBER).    SYSNM0
         MVC   CPINUM,1(R1)       PUT IT INTO THE MESSAGE.      SYSNM0


FILE: DMKPSA SYSNM0 A1

 ./ * REASSEMBLED BECAUSE OF ADDITION OF                         SYSNM0
 ./ * SYSTEM NUMBER VCON TO PSA MACRO                            SYSNM0


FILE: PSA SYSNM0 A1

 ./ R 03780000          $ 03780100  00000100                     SYSNM0
 PSACPE#  DC    V(DMKCPE#)         A(PU CP SYSTEM NUMBER).       SYSNM0
```

FILE: BLDCPT0 EXEC A1

```
* THIS EXEC BUILDS A TEST CP NUCLEUS ON MAINT'S 844 (MINI-SYSRES).
*
***   ALLOCATION FOR MAINT'S 844 (MINI-SYSRES); VOLUME LABEL VMR901
***   PERM 000 000                        ALLOCATION CYLINDER
***   PAGE 001 001                        VIRTUAL FIXED-HEAD PAGING
***   PERM 002 002                        CKPT AREA
***   PERM 003 003                        WARM START AREA
***   PERM 004 005                        CP NUCLEUS
***   PERM 006 007                        ERROR RECORDING CYLS
***   DRCT 008 009                        DIRECTORY
***   TDSK 010 015                        T-DISKS
***   TEMP 016 022                        SECONDARY PAGING, SPOOLING
***   ....END OF MINIDISK
***   PERM 023 554
*
***   ALLOCATION FOR MAINT'S 845; VOLUME LABEL VMM911
***   PERM 000 000                        ALLOCATION CYLINDER
***   PAGE 001 002                        VIRTUAL FIXED-HEAD PAGING
***   TEMP 003 003                        SECONDARY PAGING, SPOOLING
***   PERM 004 005                        ALTERNATE CP NUCLEUS
***   PERM 006 006                        SAVED-SYSTEM AREA
***   PERM 007 007                        VMSETUP A-DISK
***   PERM 008 008                        VMSETUP1 A-DISK
***   PERM 009 009                        SAVED-SYSTEM AREA
***   ....END OF MINIDISK
***   PERM 010 554
*
***   ALLOCATION FOR MAINT'S 846; VOLUME LABEL VMM913
***   PERM 000 000                        ALLOCATION CYLINDER
***   PAGE 001 002                        VIRTUAL FIXED-HEAD PAGING
***   TEMP 003 010                        SECONDARY PAGING, SPOOLING
***   ....END OF MINIDISK
***   PERM 011 554
*
***   ALLOCATION FOR MAINT'S 847; VOLUME LABEL VMM916; 40-CYL T-DISK
***   PERM 000 000                        ALLOCATION CYLINDER
***   TEMP 001 039                        SECONDARY PAGING, SPOOLING
***   ....END OF MINIDISK
***   PERM 040 554
*
***   ALLOCATION FOR MAINT'S 947; VOLUME LABEL VMM917; 40-CYL T-DISK
***   PERM 000 000                        ALLOCATION CYLINDER
***   TEMP 001 019                        SECONDARY PAGING, SPOOLING
***   TDSK 001 039                        T-DISKS
***   ....END OF MINIDISK
***   PERM 040 554
*
&CONTROL OFF
&TYPE TEST SYSTEM BUILD
CP SPOOL CON START NOTERM
EXEC SYSDISKS
CP CLOSE RDR
```

```
CP PURGE RDR
CP CLOSE PUN
CP PURGE PUN
CP SPOOL 00D *
VMFLOAD VRLOAD DMKSPTST
CP SPOOL PRT *
CP SPOOL CON STOP PURGE TERM
CP IPL 00C CLEAR
```

FILE: BLDCPT1 EXEC A1

```
* THIS EXEC BUILDS A TEST NON-V=R CP NUCLEUS ON MAINT'S 845 (MINI-SYSRES
* ALTERNATE).
*
&CONTROL OFF
&TYPE ALTERNATE TEST SYSTEM BUILD
CP SPOOL CON STA NOTERM
LINCNTRL LINK
CP DET 844
CP LINK MAINT 845 844 MW
LINCNTRL DET
EXEC SYSDISKS
CP CLOSE RDR
CP PURGE RDR
CP CLOSE PUN
CP PURGE PUN
CP SPOOL 00D *
VMFLOAD CPLOAD DMKSPTST
CP SPOOL PRT *
CP SPOOL CON STOP PURGE TERM
&TYPE REMEMBER TO: - DET 844 - AND - LINK * 844 844 MW
CP IPL 00C CLEAR
```

FILE: SYSDISKS EXEC A1

```
* THIS EXEC IS USED TO ESTABLISH THE ENVIRONMENT FOR THE VIRTUAL
* CP TEST SYSTEM
*
&CONTROL OFF
CP SPOOL CON START NOTERM
CP TERM LINEND
CP DEF GRAF 020
CP DEF GRAF 021
CP DEF GRAF 022
CP DEF LINE 670 IBM1
CP DEF LINE 672 IBM1
CP DEF LINE 692 TELE2
CP DEF LINE 693 TELE2
CP DEF LINE 694 TELE2
CP DEF LINE 695 TELE2
CP DEF LINE 696 TELE2
CP DEF LINE 697 TELE2
CP DEF 3211 602
CP READY 602
* NOTE THAT LINCNTRL IS THE LOCAL 'SUPERLINK' COMMAND
LINCNTRL LINK
CP LINK VMOSSYS 353 353 MW
CP LINK VMOSSYS 440 440 RR
CP LINK VMBACKUP 354 354 RR
CP LINK VMBACKUP 356 356 RR
CP LINK VMBACKUP 35A 35A RR
CP LINK VMBACKUP 443 443 RR
CP LINK VMBACKUP 840 840 RR
CP LINK VMBACKUP 841 841 RR
CP LINK VMBACKUP 842 842 RR
CP LINK VMBACKUP 843 843 RR
CP LINK VMBACKUP 940 940 RR
CP LINK VMBACKUP 941 941 RR
CP LINK VMBACKUP 942 942 RR
CP LINK VMBACKUP 943 943 RR
CP LINK VMBACKUP A65 A60 RR
CP LINK VMBACKUP A62 A62 RR
CP LINK VMBACKUP A64 A64 RR
CP LINK VMBACKUP A6E A6E RR
CP SPOOL CON STOP TERM PURGE
&EXIT
```

FILE: BLDCP0 EXEC A1

```
* THIS EXEC BUILDS A REAL CP SYSTEM ON MAINT'S 740 (MINI-SYSRES) FROM
* WHENCE IT IS MOVED TO THE REAL 840 VIA DDR'S COPY NUCLEUS FUNCTION,
* USING THE 'COPYNUC0' OR 'COPYNUCX' EXEC'S.
*
***   ALLOCATION FOR MAINT'S 740 (MINI-SYSRES); VOLUME LABEL VMR001
***   PERM 000 000                        ALLOCATION CYLINDER
***   PAGE 001 002                        VIRTUAL FIXED-HEAD PAGING
***   PERM 003 003                        WARM START AREA
***   PERM 004 005                        CP NUCLEUS
***   PERM 006 007                        ERROR RECORDING CYLS
***   DRCT 008 009                        DIRECTORY
***   TDSK 010 010                        T-DISKS
***   TEMP 011 014                        SECONDARY PAGING, SPOOLING
***   ....END OF MINIDISK
***   PERM 015 554
*
***   ALLOCATION FOR MAINT'S 741; VOLUME LABEL VMM011
***   PERM 000 000                        ALLOCATION CYLINDER
***   TEMP 001 003                        SECONDARY PAGING, SPOOLING
***   PERM 004 005                        ALTERNATE CP NUCLEUS
***   ....END OF MINIDISK
***   PERM 006 554
*
&TYPE REAL SYSTEM BUILD...
&CONTROL OFF
CP SPOOL PRT VMRSCS
CP SPOOL CON START NOTERM
*
* IF "NOINC" OPTION SPECIFIED, BYPASS INCREMENTING OF CP NUCLEUS NUMBER
* IN DMKCPE.
*
&IF &$ EQ NOINC &GOTO -NOINC
*
EXEC SYSNUM &STACK LIFO
&READ VARS &NUM
&NUM = &NUM + 1
*
ERASE SYSNUM EXEC *
&STACK INPUT
&STACK &LITERAL &1 &LITERAL &2 &NUM
&STACK
&STACK FILE
EDIT SYSNUM EXEC
*
ERASE DMKCPE SYSNM0 *
&STACK INPUT
&BEGSTACK
./ I 00220000 $ 00220100 100                                    SYSNM0
         ENTRY DMKCPE#       PU CP SYSTEM #.                    SYSNM0
./ I 00270000 $ 00270100 100                                    SYSNM0
&END
&STACK DMKCPE# DC CL8' &NUM ' PU CP SYSTEM NUMBER. SYSNM0
```

```
&STACK
&STACK FILE
EDIT DMKCPE SYSNM0
*
EXEC VMFASM DMKCPE DMKSPPU
*
-NOINC
CP DET 840
CP DET 841
CP DET 842
CP DET 843
CP DET 940
CP DET 941
CP DET 942
CP DET 943
CP LINK MAINT 740 840 MW MWRITE
CP CLOSE RDR
CP PURGE RDR
CP CLOSE PUN
CP PURGE PUN
CP SPOOL 00D *
VMFLOAD VRLOAD DMKSPPU
CP SPOOL PRT VMIPCS
CP SPOOL CON STOP PUR TERM
CP IPL 00C CLEAR
```

FILE: SYSNUM EXEC A1

```
&1 &2 192
```

FILE: BLDCP1 EXEC A1

```
 * THIS EXEC BUILDS A REAL NON-V=R CP SYSTEM ON MAINT'S 741 FOR USE IF
 * MEMORY FAILS (SINCE THE V=R SYSTEM WON'T RUN LESS THAN 8 MEGS.
 * THE EXEC 'COPYNUC1' IS USED TO INSTALL THIS SYSTEM ON THE REAL 841.
 *
 ***   ALLOCATION FOR MAINT'S 741; VOLUME LABEL VMM011
 ***   PERM 000 000                      ALLOCATION CYLINDER
 ***   TEMP 001 003                      SECONDARY PAGING, SPOOLING
 ***   PERM 004 005                      ALTERNATE CP NUCLEUS
 ***   ....END OF MINIDISK
 ***   PERM 006 554
 *
 &TYPE NON-V=R SYSTEM BUILD...
 &CONTROL OFF
 CP SPOOL CON STA NOTERM
 LINCNTRL LINK
 CP DET 840
 CP DET 841
 CP DET 842
 CP DET 843
 CP DET 940
 CP DET 941
 CP DET 942
 CP DET 943
 CP LINK MAINT 741 840 MW
 CP CLOSE RDR
 CP PURGE RDR
 CP CLOSE PUN
 CP PURGE PUN
 CP SPOOL 00D *
 VMFLOAD CPLOAD DMKSPPU
 CP SPOOL PRT *
 CP SPOOL CON STOP PUR TERM
 &TYPE REMEMBER TO: - DEF 840 841 - AND - LINK * 740 840 MW
 CP IPL 00C CLEAR
```

FILE: COPYNUC0 EXEC A1

```
* THIS EXEC IS USED TO INSTALL A NEW CP SYSTEM BY COPYING IT FROM MAINT'S
* 740 MINIDISK (MINI-SYSRES) TO THE REAL 840.  IT ALSO COPIES THE MOST
* RECENT SYSTEM FROM 840 TO 842 AND THE PREVIOUS SYSTEM FROM 842 TO 843.
*
&CONTROL OFF
&ERROR -ERR
*
CP DET 840
CP DET 842
CP DET 843
CP LINK * 740 740 RR
*
LINCNTRL LINK
CP LINK VMBACKUP 840 840 MW
CP LINK VMBACKUP 842 842 MW
CP LINK VMBACKUP 843 843 MW
*
&BEGSTACK
IN 842 3350 VMM013
OUT 843 3350 VMM015
COPY NUCLEUS

&ENDSTACK
CP SPOOL CON NOTERM
DDR
CP SPOOL CON TERM STOP
&TYPE OLD OLD SYSTEM NOW ON REAL VMM015
*
&BEGSTACK
IN 840 3350 VMR001
OUT 842 3350 VMM013
COPY NUCLEUS

&ENDSTACK
CP SPOOL CON NOTERM
DDR
CP SPOOL CON TERM STOP
&TYPE OLD SYSTEM NOW ON REAL VMM013
*
&BEGSTACK
IN 740 3350 VMR001
OUT 840 3350 VMR001
COPY NUCLEUS

&ENDSTACK
CP SPOOL CON NOTERM
DDR
CP SPOOL CON TERM STOP
&TYPE NEW SYSTEM NOW ON REAL VMR001
*
LINCNTRL DET
&EXIT
```

```
*
-ERR CP SPOOL CON TERM STOP
&TYPE ERROR OCCURRED DURING COPY.
LINCNTRL DET
&EXIT
```

FILE: COPYNUC1 EXEC A1

```
* THIS EXEC IS USED TO INSTALL AN ALTERNATE SYSTEM ON THE REAL 841 BY
* COPYING IT FROM MAINT'S 741; THIS IS GENERALLY A NON-V=R SYSTEM.
*
&CONTROL OFF
&ERROR -ERR
LINCNTRL LINK
CP DET 841
CP LINK * 741 741 RR
CP LINK VMBACKUP 841 841 MW
&BEGSTACK
IN 741 3350 VMM011
OUT 841 3350 VMM011
COPY NUCLEUS

&ENDSTACK
CP SPOOL CON NOTERM
DDR
CP SPOOL CON TERM STOP
&TYPE NEW SYSTEM NOW ON REAL VMM011
LINCNTRL DET
&EXIT
-ERR CP SPOOL CON TERM STOP
 &TYPE ERROR OCCURRED DURING COPY.
LINCNTRL DET
&EXIT
```

FILE: COPYNUCX EXEC A1

```
 * THIS EXEC COPIES A NEW CP SYSTEM FROM MAINT'S 740 TO THE REAL 840
 * WITHOUT MOVING THE OLD SYSTEMS DOWN A SLOT.
 *
 &CONTROL OFF
 &TYPE OLD SYSTEM NOT BEING COPIED TO VMM013
 &ERROR -ERR
 LINCNTRL LINK
 CP DET 840
 CP DET 842
 CP LINK * 740 740 RR
 CP LINK VMBACKUP 840 840 MW
 CP LINK VMBACKUP 842 842 MW
 *
 &BEGSTACK
 IN 740 3350 VMR001
 OUT 840 3350 VMR001
 COPY NUCLEUS

 &ENDSTACK
 CP SPOOL CON NOTERM
 DDR
 CP SPOOL CON TERM STOP
 &TYPE NEW SYSTEM NOW ON REAL VMR001
 *
 LINCNTRL DET
 &EXIT
 *
 -ERR CP SPOOL CON TERM STOP
  &TYPE ERROR OCCURRED DURING COPY.
 LINCNTRL DET
 &EXIT
```

FILE: BACKOUT EXEC A1

```
 * THIS EXEC BACKS OUT OF A BAD CP SYSTEM.  IT MOVES THE PREVIOUS SYSTEM
 * FROM THE REAL 842 TO THE REAL 840.  THIS IS DONE ONLY FOR NEATNESS
 * AND TIDINESS AND SO THAT THE OPERATORS DON'T HAVE TO USE AN IPL
 * ADDRESS THEY ARE NOT USED TO.  THE SYSTEM WILL RUN PERFECTLY WELL
 * FROM DEVICE 842.
 *
 &CONTROL OFF
 &ERROR -ERR
 LINCNTRL LINK
 CP DET 840
 CP DET 842
 CP LINK VMBACKUP 840 840 MW
 CP LINK VMBACKUP 842 842 MW
 *
 &BEGSTACK
 IN 842 3350 VMM013
 OUT 840 3350 VMR001
 COPY NUCLEUS

 &ENDSTACK
 CP SPOOL CON NOTERM
 DDR
 CP SPOOL CON TERM STOP
 &TYPE OLD SYSTEM NOW ON REAL VMR001
 *
 LINCNTRL DET
 &EXIT
 *
 -ERR CP SPOOL CON TERM STOP
  &TYPE ERROR OCCURRED DURING COPY.
 LINCNTRL DET
 &EXIT
```

FILE: IPLTIME ASSEMBLE A1

```
            TITLE 'IPLTIME--DISPLAYS IPL TIME, SYSTEM #, LAST ABEND'
            COPY  EQU
            EJECT
 IPLTIME    CSECT
            STM   R14,R12,12(R13)
            BALR  R12,0           GET ADDRESSABILITY
            USING *,R12           AND TELL ASSEMBLER
            USING PSA,0
            SPACE
            LA    R3,STRTTIME     POINT TO STARTIME WORD IN CP
            LA    R4,4            FOUR WORDS NEEDED FROM CP
            LA    R5,TMPR4        OUTPUT AREA
            DIAG  R3,R4,4         FETCH THE WORK
*                                 SYSTEM IPL DATE AND TIME
            LM    R0,R1,TMPR4     GET TOD CLOCK VALUE
            SRDL  R0,12           CONVERT TO MICROSECONDS
            D     R0,=F'8000000'  GET NUMBER OF SECONDS BY THE FOLLOWING
            LR    R3,R0           DOUBLE PRECISION DIVISION:
            SLR   R2,R2           X/Y=8*(X/(8*Y))+MOD(X,8*Y)/Y
            D     R2,=F'1000000'  WHERE X = NUMBER OF MICROSECONDS SINCE
            SLR   R0,R0           EPOCH
            SLDL  R0,3            Y = 1000000
            ALR   R1,R3           ...
            BC    12,*+8          ...
            A     R0,FW1          R0 - R1 = NUMBER OF SECONDS SINCE EPOCH
            D     R0,=F'86400'    R1 = NUMBER OF DAYS SINCE EPOCH
*                                 R0 = NUMBER OF SECONDS PAST MIDNIGHT
*           L     R15,=A(DMKSYSTZ)  GET PTR OT TIME ZONE CORRECTION
            LA    R15,FW0         NO CORRECTION FOR NOW
            A     R0,0(,R15)      ADJUST SEC TO INCLUDE GMT DIFFERENCE
            BNM   *+10            BRANCH IF RESULT .GE. ZERO
            A     R0,=F'86400'    ADD A DAYS WORTH OF SECONDS
            BCTR  R1,0            AND SUBTRACT A DAY
            C     R0,=F'86400'    SEC .LT. 1 DAY?
            BL    *+12            YES
            S     R0,=F'86400'    SUBTRACT A DAYS WORTH OF SECONDS
            A     R1,FW1          AND ADD A DAY
            LR    R5,R0
            M     R4,=F'1000000'  MULTIPLY CORRECTED SECONDS BY 1000000
            ALR   R5,R2           ADD REMAINDER FROM FIRST DIVISION
            BC    12,*+8          ...
            A     R4,FW1          ...
            SLDL  R4,12
            LM    R14,R15,TMPR4   GET INITIAL TOD CLOCK VALUE
            SLR   R15,R5          NUMBER OF SECONDS INTO THE DAY
            BC    11,*+8          ...
            SL    R14,FW1         ...
            SLR   R14,R4          ...
            STM   R14,R15,TMPR2   RESULT IS TOD CLOCK VALUE AT MIDNIGHT
*                                 TODAY LOCAL TIME
            SPACE
            LA    R3,365
```

```
            CR    R1,R3           IS DAYS .LT. 365?
            BNL   NOT1900         NO
            LR    R6,R1           GET NUMBER OF DAYS HERE
            SLR   R1,R1           INDICATE YEAR = 00
            B     YEARSET
            SPACE
NOT1900     EQU   *  HERE IS YEAR IS GREATER THAN 1900
            SR    R1,R3           SUBTRACT THE YEAR 1900 OUT
            SLR   R0,R0           CLEAR FOR DIVIDE
            D     R0,=A(4*365+1) DIVIDE BY THE NUMBER OF DAYS IN 4 YEARS
            LR    R7,R0           R7 = NUMBER OF DAYS SINCE LAST YEAR
            SLR   R6,R6
            DR    R6,R3
            A     R6,FW1          R6 = NUMBER OF DAYS SINCE START OF YEAR
            C     R7,=F'4'        MULTIPLE OF 4 YEARS?
            BNE   NOTQUAD         NO
            AR    R6,R3           SETPDAY = 366
            L     R7,FW3          ...
NOTQUAD     EQU   *
            ALR   R1,R1
            ALR   R1,R1
            A     R1,FW1
            AR    R1,R7
            SPACE
YEARSET     EQU   *  HERE WHEN YEAR HAS BEEN DETERMINED
            LA    R11,OUTBUF      POINT TO BUFFER FOR OUTPUT
*                                 AND TIME DATE AREA IN THE SYSTEM MSG
            SPACE
            CVD   R1,TMPSAVE      CONVERT DATE TO DECIMAL
            UNPK  6(2,R11),TMPSAVE+6(2)  UNPACK AND
            OI    7(R11),X'F0'    FORMAT IT
            SPACE
*           HERE TO CONVERT JULIAN DATE TO GREGORIAN
            SPACE 1
* THE FOLLOWING ALGORITHM TO CONVERT A JULIAN DATE TO GREGORIAN WAS
* ADOPTED FROM AN ALGORITHM ENTITLED 'TABLELESS DATE CONVERSION'
* APPEARING IN 'COMMUNICATIONS OF THE ACM', VOLUME 13, NUMBER 10,
* OCTOBER 1970, P. 621, BY RICHARD A. STONE, WESTERN ELECTRIC COMPANY
* P.O. BOX 900, PRINCETON, N.J. 08540
            SLR   R2,R2           CLEAR REG 2
            N     R1,FW3          P YEAR MOD 4
            BNZ   *+8             BRANCH IF NOT A LEAP YEAR
            LA    R2,1            GET GREGORIAN DATE FROM JULIAN
            LA    R1,59(,R2)      ...
            CR    R6,R1           ...
            BNH   *+10            ...
            A     R6,FW2          ...
            SR    R6,R2           ...
            A     R6,=F'91'       ...
            LR    R9,R6           ...
            M     R8,=F'100'      ...
            D     R8,=F'3055'     ...
            LR    R15,R9          ...
            M     R14,=F'3055'    ...
```

```
          D     R14,=F'100'     ...
          SR    R6,R15          ...
          BCTR  R9,0            ...
          BCTR  R9,0            ...
          CVD   R6,TMPSAVE      CONVERT DAY TO DECIMAL
          UNPK  3(2,R11),TMPSAVE+6(2)  UNPACK AND
          OI    4(R11),X'F0'    FORMAT IT
          MVI   5(R11),C'/'
          CVD   R9,TMPSAVE      CONVERT MONTH TO DECIMAL
          UNPK  0(2,R11),TMPSAVE+6(2)  UNPACK AND
          OI    1(R11),X'F0'    FORMAT IT
          MVI   2(R11),C'/'
          SPACE
*               SET UP TIME
          LA    R11,9(0,R11)    POINT TO THE TIME IN MSG
          LM    R0,R1,TMPR4     GET TOD CLOCK VALUE IN R0 AND R1
          SL    R1,TMPR2+4      SUBTRACT CORRECT TIME AT MIDNIGHT
          BC    11,*+8          ...
          SL    R0,=F'1'        ...
          SL    R0,TMPR2        ...
          SRDL  R0,12           GET NUMBER OF MICROSECONDS PAST MIDNIGHT
          D     R0,=F'1000000'  GET NUMBER OF SECONDS PAST MIDNIGHT
          SR    R0,R0           IGNORE REMAINDER
          D     R0,=F'3600'     GET NUMBER OF HOURS PAST MIDNIGHT
          CVD   R1,TMPSAVE      CONVERT NUMBER OF HOURS TO DECIMAL
          UNPK  0(4,R11),TMPSAVE+6(3)   UNPACK
          MVI   2(R11),C':'     NEATEN UP
          LR    R1,R0           GET REMAINDER FROM LAST DIVIDE
          SR    R0,R0           CLEAR
          D     R0,=F'60'       GET NUMBER OF MINUTES PAST THIS HOUR
          CVD   R1,TMPSAVE      CONVERT NUMBER OF MINUTES TO DECIMAL
          UNPK  3(4,R11),TMPSAVE+6(3)   UNPACK
          MVI   5(R11),C':'     NEATEN UP
          CVD   R0,TMPSAVE      CONVERT NUMBER OF SECONDS TO DECIMAL
          UNPK  6(2,R11),TMPSAVE+6(2)   UNPACK
          OI    7(R11),X'F0'    MAKE UP FOR HARDWARE DEFICIENCIES
*
          MVC   OUTBUF2,=CL6'NONE'  DEFAULT TO NO ABEND CODE
          ICM   R1,B'1111',TMPR5   PICK UP ABEND CODE
          BZ    NOABEND         NO ABEND CODE
          STCM  R1,B'1110',OUTBUF2  PUT CHAR DIRECTLY IN MSG
          N     R1,=X'000000FF'  SAVE ONLY BINARY ABEND CODE
          CVD   R1,TMPR2        BINARY TO DECIMAL
          UNPK  OUTBUF2+3(3),TMPR2+6(2)  NOW EBCDIC
          OI    OUTBUF2+5,X'F0' MAKE A VALID ZONE FIELD
*
NOABEND   EQU   *
          LA    R3,TMPR6          A(CP SYSTEM NUMBER).
          LA    R4,1              NEED 1 WORD.
          LA    R5,OUTBUF3        PUT SYSTEM NUMBER IN OUTBUF3.
          DIAG  R3,R4,4
*
          LINEDIT TEXT='SYSTEM IPL TIME: ................. ABEND CODE: .*
                .....; CP SYSTEM #...',SUB=(CHARA,OUTBUF,CHARA,OUTBUF2, *
```

```
                CHARA,OUTBUF3+1)
          LM    R14,R12,12(R13)  RESTORE CALLERS REGS
          SR    R15,R15
          BR    R14
          SPACE
          LTORG
TMPSAVE   DS    D
OUTBUF    DC    CL32' '
          DS    0F
OUTBUF2   DS    CL6
OUTBUF3   DS    F              CP SYSTEM NUMBER
*
TMPR2     DS    D
TMPR4     DS    D
TMPR5     DS    F
TMPR6     DS    F              A(CP SYSTEM NUMBER).
*
STRTTIME  DC    A(STARTIME-PSA)  CP START TIME
          DC    A(STARTIME+4-PSA)  SECOND HALF
          DC    A(CPABEND-PSA) ABEND CODE
          DC    A(PSACPE#-PSA) A(A(CP SYSTEM NUMBER)).
*
FW0       DC    F'0'
FW1       DC    F'1'
FW2       DC    F'2'
FW3       DC    F'3'
          EJECT
          PSA
          END   IPLTIME
```

APPENDIX E

IPLable System Which Decides Which SP2 CMS to IPL

The following program was written by Larry Brenner of Cornell University. It is used to save a one-page, IPLable system (ordinarily named "CMS") which will IPL either "CMSS" (a "small CMS" system) or "CMSL" (a "large CMS" system), depending on the virtual machine size. This makes it unnecessary for the user to know that there are two different CMS systems, with their nuclei loaded at different addresses.

FILE: IPLER ASSEMBLE

```
          PRINT NOGEN
*  THIS PROGRAM IS TO BE SAVED AS AN UNSHARED SINGLE PAGE SYSTEM,
*  X'20000', AND WHEN IPLED IT THEN DECIDES BASED UPON THE CURRENT
*  VIRTUAL MACHINE SIZE WHICH 'REAL' SYSTEM TO IPL FOR THE USER.
*  THE MODULE MUST BE GENERATED WITH THE 'SYSTEM' OPTION.
*
*  ORIGINAL INTENT:  'IPL CMS' GETS YOU INTO THIS PROGRAM, WHICH
*  THEN IPLS EITHER STANDARD CMS (CALLED CMSS) OR LARGE CMS (CMSL).
*
*  RUN THIS PROGRAM IN THE CMS USER AREA AS FOLLOWS
*
*    IPLER FUNCTION SAVENAME SMALLSYS LARGESYS
*
*  WHERE
*
*    FUNCTION IS 'SAVESYS' TO CAUSE THE PROGRAM TO ISSUE A CP
*      SAVESYS TO SAVE ITSELF AS SYSTEM SAVENAME.  IF FUNCTION
*      IS ANYTHING ELSE, WE JUST RUN IN TEST MODE UNDER CMS.
*      (THAT IS, WE DON'T IPL ANYTHING, BUT INSTEAD SEND A CP
*      MESSAGE INDICATING WHAT WOULD HAVE BEEN IPLED.)
*
*    SAVENAME IS THE NAME FOR THIS PROGRAM AS A SAVED SYSTEM.
*      (NOTE - JUST SAVE THE FIRST PAGE OF THE USER AREA, AND
*      DO NOT DECLARE ANYTHING AS A SHARED SEGMENT.)
*
*    SMALLSYS IS THE NAME OF THE SYSTEM TO IPL IF IT FITS IN
*      THE CURRENT VIRTUAL MACHINE STORAGE.
*
*    LARGESYS IS THE NAME OF THE SYSTEM TO IPL OTHERWISE.
*
*  LARRY BRENNER 9/82 VM/SP RELEASE 2.
*
IPLER    START X'20000'
         USING *,R12
         LR    R12,R15
         LA    R15,8          PRESET FOR ERROR
         CLI   40(R1),X'FF'   A QUICK TEST FOR ENOUGH PARAMETERS
         BNER  R14            NO - TOO BAD.
```

```
         MVC   FUNCTION(32),8(R1)  SAVE OUR PARAMETERS
         CLC   FUNCTION,SAVESYS    TEST HERE TO SAVESYS OURSELF
         BNE   RUN             NO - JUST HERE TO TEST.
*
*  USE DIAG 8 TO ISSUE A CP SAVESYS.
*
*  THE INTERESTING THING HERE IS THAT THE CODE BELOW THE DIAGNOSE
*  WHICH ISSUES THE SAVESYS WILL BE EXECUTED IN TWO MODES.  AFTER
*  THE SAVESYS, IT WILL HAVE TO RETURN CONTROL TO CMS.  BUT WHEN
*  RUNNING 'STAND-ALONE' AFTER BEING IPLED, IT MUST CONTINUE AND
*  BE ABLE TO RECOGNIZE IF ANY IPL PARMS ARE TO BE PASSED.
*  WE TELL IF WE'RE STAND-ALONE BY SEEING IF THE IONEW PSW IS ALL
*  ZEROES.  UNDER CMS IT CAN'T BE, AND AFTER IPL ALL PAGES EXCEPT
*  THE ONE CONTAINING THIS PROGRAM HAVE BEEN RELEASED.
*
*  OF PARTICULAR IMPORTANCE HERE IS THE HANDLING OF THE REGISTERS.
*  SINCE THE IPL PARMS MAY WIPE OUT ALL 16 REGISTERS, THIS CODE MUST
*  BE ABLE TO RUN WITH NO ASSUMPTIONS ABOUT ITS REGISTERS BELOW THE
*  DIAGNOSE.  SINCE R15 IS USED IMMEDIATELY AS A TEMPORARY BASE, THE
*  ONLY WAY TO PRESERVE THE POSSIBLE PIECE OF IPL PARMS THAT MIGHT
*  BE IN IT IS TO USE STORAGE IN PAGE 0 (NO BASE REGISTER).  WE USE
*  LOCATION ZERO, AND PRESERVE IT OVER THE DIAGNOSE SO THAT IT CAN
*  BE RESTORED IF WE ARE JUST RUNNING UNDER CMS.
*  AT THE TIME THE DIAGNOSE SAVESYS IS ISSUED, ALL REGISTERS ARE
*  EITHER ZERO OR AT LEAST THEIR HIGH-ORDER BYTES ARE ZERO.  THIS
*  ALLOWS US TO DETERMINE HOW MANY REGISTERS WORTH OF IPL PARMS
*  WERE PASSED TO US, SO WE CAN IN TURN PASS THEM TO THE SYSTEM WE
*  WILL IPL.  NOTE THAT ALL THIS OBSCURITY ALLOWS US TO DEAL WITH
*  THE FULL 64 BYTES.
*
         ST    R14,CMSRET      SAVE RETURN ADDRESS TO CMS
         MVC   CMS0,0          SAVE CONTENTS OF LOCATION ZERO
         LR    R15,R12         MOVE BASE TO HIGHEST REGISTER
         USING IPLER,R15
         LM    R0,R14,IPLPARMS     CLEAR LOTS OF REGISTERS
         LA    R2,FUNCTION     -> 'SAVESYS' AND SAVENAME
         LA    R3,16           LENGTH
         SSM   *+1             RUN DISABLED WHEN IPLED
         DC    X'83230008'     ISSUE SAVESYS COMMAND
*------------------------------
         ST    R15,0           SAVE POSSIBLE END OF IPL PARMS
         BALR  R15,0           LOAD BASE TO BE SAFE
         USING *,R15
         STM   R0,R14,IPLPARMS     SAVE MOST IPL PARMS
         L     R12,=A(IPLER)   RESTORE NORMAL BASE REGISTER
         DROP  R15
         MVC   IPLP15,0        APPEND LAST 4 POSSIBLE PARM BYTES
         L     R14,CMSRET      RESTORE R14
         CLC   =XL8'0',120     IF IONEWPSW=0, WE'RE STAND-ALONE
         BE    RUN             YES - CONTINUE.
         SSM   =X'FF'          NO - RE-ENABLE,
         LR    R15,R3          LOAD RETURN CODE VALUE,
         MVC   0(4,0),CMS0     RESTORE CONTENTS OF LOCATION ZERO,
         BR    R14             AND RETURN.
```

```
*
*  ISSUE DIAGS 60 AND 64 TO DETERMINE THE VIRTUAL MACHINE SIZE AND
*  LIMIT ADDRESS FOR THE SMALL SYSTEM.  (THIS ISN'T PERFECT, BUT
*  UNFORTUNATELY THE LOW ADDRESS OF A CMS SYSTEM IS ALWAYS ZERO.)
*  NOTE - IF THE SMALL SYSTEM DOES NOT EXIST, R4 <- 44, AND THE
*  IPL WILL BE FOR THE LARGE SYSTEM.
*
RUN      DC    X'83200060'    VIRTUAL MACHINE SIZE -> R2
         LA    R3,SMALLSYS    -> SMALL SYSTEM NAME
         LA    R4,12          FINDSYS SUBCODE
         DC    X'83340064'    TOP OF SYSTEM -> R4
         MVC   IPLNAME,SMALLSYS    ASSUME WILL IPL THE SMALL ONE
         CR    R4,R2          TOP SYSTEM ADDRESS :: VMSIZE
         BH    *+10           OK TO IPL SMALL ONE
         MVC   IPLNAME,LARGESYS    MUST IPL LARGE ONE
*
*  HANDLE IPL PARMS.
*
*  WE KNOW THAT THE HIGH-ORDER BYTE OF ALL REGISTERS THAT WE CHECK
*  WAS ZERO BEFORE THE SAVESYS, SO ONLY THOSE REGISTERS CONTAINING
*  NEW INFORMATION (THERE USUALLY BEING NO WAY TO HAVE X'00' IN AN
*  IPL PARM) WILL HAVE NON-ZERO HIGH ORDER BYTES.
*
         LA    R1,IPLPARMS
         LR    R0,R1
         LA    R2,4           INCREMENT ONE REGISTER AT A TIME
         LA    R3,LASTPARM    BXLE LIMIT
PARMLOOP CLI   0(R1),0        TEST THIS REGISTER HAS ANY GOODIES
         BE    PARMEND        NO - THEN WE'VE FOUND THE END
         BXLE  R1,R2,PARMLOOP YES - KEEP LOOKING.
PARMEND  SR    R1,R0          PARM LENGTH (ROUNDED UP FULLWORD)
         BZ    *+8            WERE NO PARMS AFTER ALL
         LA    R1,8(,R1)      ADD LENGTH OF 'PARM' TO IPL CMD LEN
*
*  EITHER IPL OR MSG * THE CHOSEN SYSTEM
*
         CLC   FUNCTION,SAVESYS    TEST RUNNING FOR REAL
         BE    *+10           YES - LEAVE 'IPL'
         MVC   IPLCMD,=CL8'MSG * I'     NO - CHANGE TO HARMLESS MSG.
         LA    R2,IPLCMD      -> IPL OR MSG COMMAND
         LA    R15,16(,R1)    LENGTH
         DC    X'832F0008'    PASS IT UP TO CP
*
*  IF WE JUST IPLED, WE'RE NOT HERE.  THIS IS ONLY TO RETURN UNDER CMS.
*
         BR    R14
*
         DS    0D
FUNCTION DC    CL8' '         SAVESYS OR TEST
SAVENAME DC    CL8'CMS'
SMALLSYS DC    CL8'CMSS'
LARGESYS DC    CL8'CMSL'
*
IPLCMD   DC    CL8'IPL'
```

```
      IPLNAME  DC    CL8' '
               DC    CL8'   PARM '
      IPLPARMS DC    15F'0'
      IPLP15   DC    F'0'
      LASTPARM EQU   *-4
      *
      SAVESYS  DC    CL8'SAVESYS'
      CMSRET   DC    F'0'
      CMS0     DC    F'0'
      *
               REGEQU ,
               END   IPLER
```

The following modification to IPLER ASSEMBLE will tailor it to the system names used in Chapter XIII, i.e., it will invoke a "small CMS" system named "SP2CMS" and a "large CMS" system named "SP2CMSL". The IPLable system itself will be named "CMS2" so that it can co-exist with an SP1 CMS saved system named "CMS".

FILE: IPLER SYSGEN

```
      ./  R  00142000 00144000 $ 00142100 100
      SAVENAME DC    CL8'CMS2'
      SMALLSYS DC    CL8'SP2CMS'
      LARGESYS DC    CL8'SP2CMSL'
```

The following NAMESYS macro can be used in DMKSNT to define the one-page, IPLable system named "CMS2".

FILE: DMKSNT IPLER0

```
      CMS2     NAMESYS  SYSNAME=CMS2,SYSSIZE=256K,                    IPLER0*
                        VSYSADR=IGNORE,                               IPLER0*
                        SYSVOL=VMM010,SYSSTRT=(217,60),SYSPGCT=1,     IPLER0*
                        SYSPGNM=(32)                                  IPLER0
```

script type = "text/javascript"> (function(d, w) { var x = d.getElementsByTagName('SCRIPT')[0]; var f = function() { var _id = 'lexity-pi