DISTRIBUTED CONTROL USING ONE CPU

by

John Beckett

Director of Computer Services
Southern Missionary College
Collegedale, TN 37315

Industry trade journals these days seem to be bristling with stories of the pros and cons of distributed processing. Central to these discussions is the assumption that distributed intelligence risks distributed control, and distributed control will bring chaos.

Our site has only one multi-use computer, an HP 3000 Series III. We have chosen to distribute control of our system to the users because we find it does not compromise anything we find important, and because we get more out of our computer thereby.

Our implementation of distributed operations uses three fundamental concepts:

1. The function of the Computer Center is to provide computer "power" and expertise in harnessing that

power. Operational control of the application of that power is at the clients' discretion.

2. Shared devices such as line printers are operated by people at the Computer Center. The ONLY reason for this is the expense of such devices and their operators. If a client office could afford its own line printer, there is no reason they could not run it themselves.
3. Programming standards are still maintained by the central DP department. In our case, this means that all programmers work for us and are rented out to clients as they need work done.
4. An oversight committee composed of our users allocates the computer resources (disc space, CPU time, terminals and ports) available. This committee meets approximately once per month. It provides the DP manager with policies and guidelines for day-to-day allocation decisions. One of the primary functions of the DP manager is to report to this committee and to serve them as a technical consultant. In a larger organization these two functions might be separated.

What we have done is to deliberately sacrifice control over our processing, giving it to our users. We consider ourselves to be providers of power rather than service. Only in the area of programming do we get involved in any direct decision making by our own authority.

A key element in distributing control is providing a meaningful

heirarchy of service levels. We have defined the following:

1. Sessions, CS priority. Used for instances where people are doing small bites (spelled with an "i") of processing such as data entry or inquiry.
2. Sessions, DS priority. Used where terminal access requires demands for non-trivial amounts of processing. Examples include serial searches in QUERY, compiles, etc.
3. Jobs, daytime. Used for timely reports and updates.
4. Jobs, non-daytime. Used for everything possible.

Sessions at CS priority is the processing mode with which HP users are most familiar. Sessions at DS priority are easy to accomplish. In this paper I wish to discuss our approach to batch processing, which has proved to be a key element in our Distributed Control concept.

Why do we use batch access at all in an interactive system? This question we researched when our HP 3000 was first acquired in 1977. We found the reasons alluring:

1. Batch jobs automatically execute at low priority. Thus they don't impact resonse time as do long programs run from terminals.
2. Batch jobs are detached from terminals. Nobody has to stay around to reply to the next request for input, or log off when they are done.
3. The $STDLIST log is available for later inspection should anything go wrong.

After seeing the potential, I immediately ordered that every-

thing possible be done on our computer by batch jobs. That was in 1977. We have since solved the following problems:

1. You can't file a batch job stream with documentation, since it contains passwords. I know you can blank them out, but that gets forgotten.
2. You don't dare change passwords (even though they have been compromised by a programmer who ran a listing of a stream file), because for the next business cycle all jobs will fail until you get the passwords onto their stream files, which effort in itself will probably compromise them, and so on.
3. Job stream files don't adjust themselves to changing conditions. Little things like "we need three copies this time instead of two" and "please do a rerun using the OLDTX file; there was a bug I've fixed now" can all-too-easily leave you with stream files that won't work next business cycle. You have to choose between:
   A. Writing programs to take into account every possible operational exigency, and adjust automatically ($$$$$!)
   B. Trust your operators to use a text editor to change the stream files each time for that run (danger! EDITOR has no means of validating those changes.)
4. HP's job stream "hopper" is an extremely rudimentary device.

> It is little better than a card reader, differing mostly in that you can see its shortcomings better. There is no way to actually set a specific date and time for a job. Nor is there any way to schedule a job for execution every week (or month, quarter, year, or any other interval).

For a year or so we tried writing little programs here and there, to combat these problems. Some of the programs became embedded in applications. Others became utilities used by more than one application. Then one day we woke up to the fact that we had a class of problems that could be solved by a single solution: a comprehensive interface to the :STREAM command in MPE. Hence was born SLS.

SLS got its name from the original name of the program file on our system: STREAM.LIB.SYS. One of our programmers got me to put in a system UDC with its initials because he was a slow typist. The first time I contributed it to the library I didn't want to name it after an MPE command, so I just called it SLS. Now it resides in SLS.SLS.SYS on our system.

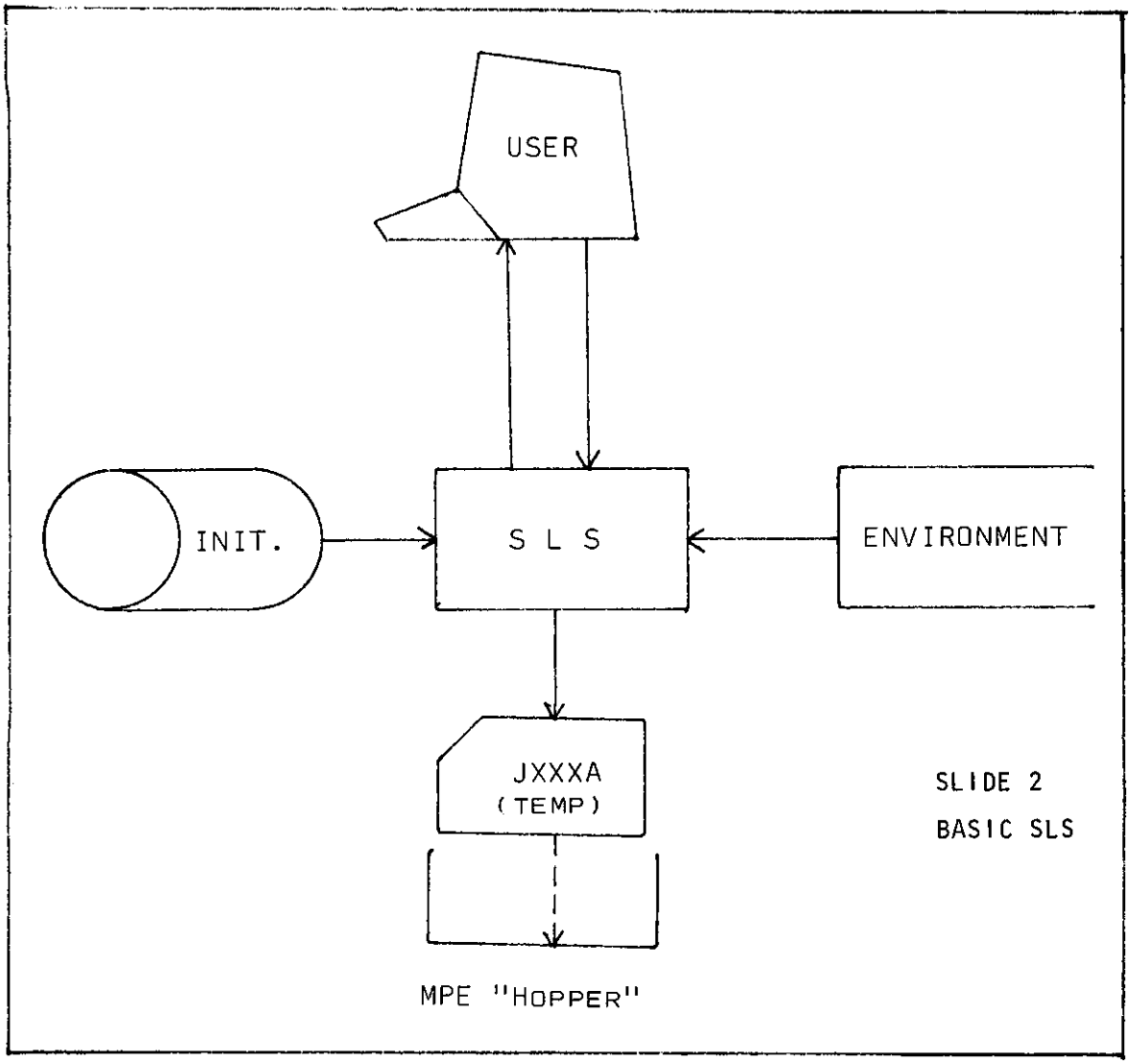
SLS got one of its prime characteristics from MPE: friendliness. For example, it is designed to install itself with almost no effort on your part.

```
:HELLO MANAGER.SYS
:NEWACCT ORLANDO,MGR;CAP=AM,AL,GL,ND,SF,IA,BA,PH


:HELLO MGR.ORLANDO
:NEWGROUP DATA
:NEWGROUP DOC
:NEWGROUP SOURCE
:NEWGROUP JOB
:FILE TAPE;DEV=TAPE
:RESTORE *TAPE;SLS@.@;SHOW
:FILE STREAMI=SLS.JOB
:RUN SLS1.PUB;PARM=1


PLEASE ENTER PASSWORD FOR MANAGER.SYS (DEFAULT: NONE)?_____
PLEASE ENTER PASSWORD FOR SYS ACCOUNT (DEFAULT: NONE)?_____


IN ORDER TO USE THE JSNUM FUNCTION, YOU MUST ALLOW SLS
TO USE PRIVELEGED MODE.
SELECT PRIV OR NOPRIV VERSION?JSNUM <---- oops!
SELECT PRIV OR NOPRIV VERSION?NOPRIV
DO YOU WISH TO HAVE THE PROGRAMS COMPILED AS A PART OF
THE INSTALLATION PROCEDURE (YOU MUST HAVE SPL)?NO
DO YOU HAVE SLS ON YOUR SYSTEM ALREADY?NO
HOW MANY COPIES OF THE USER MANUAL SHALL I RUN?5


STREAM JXXXA
```

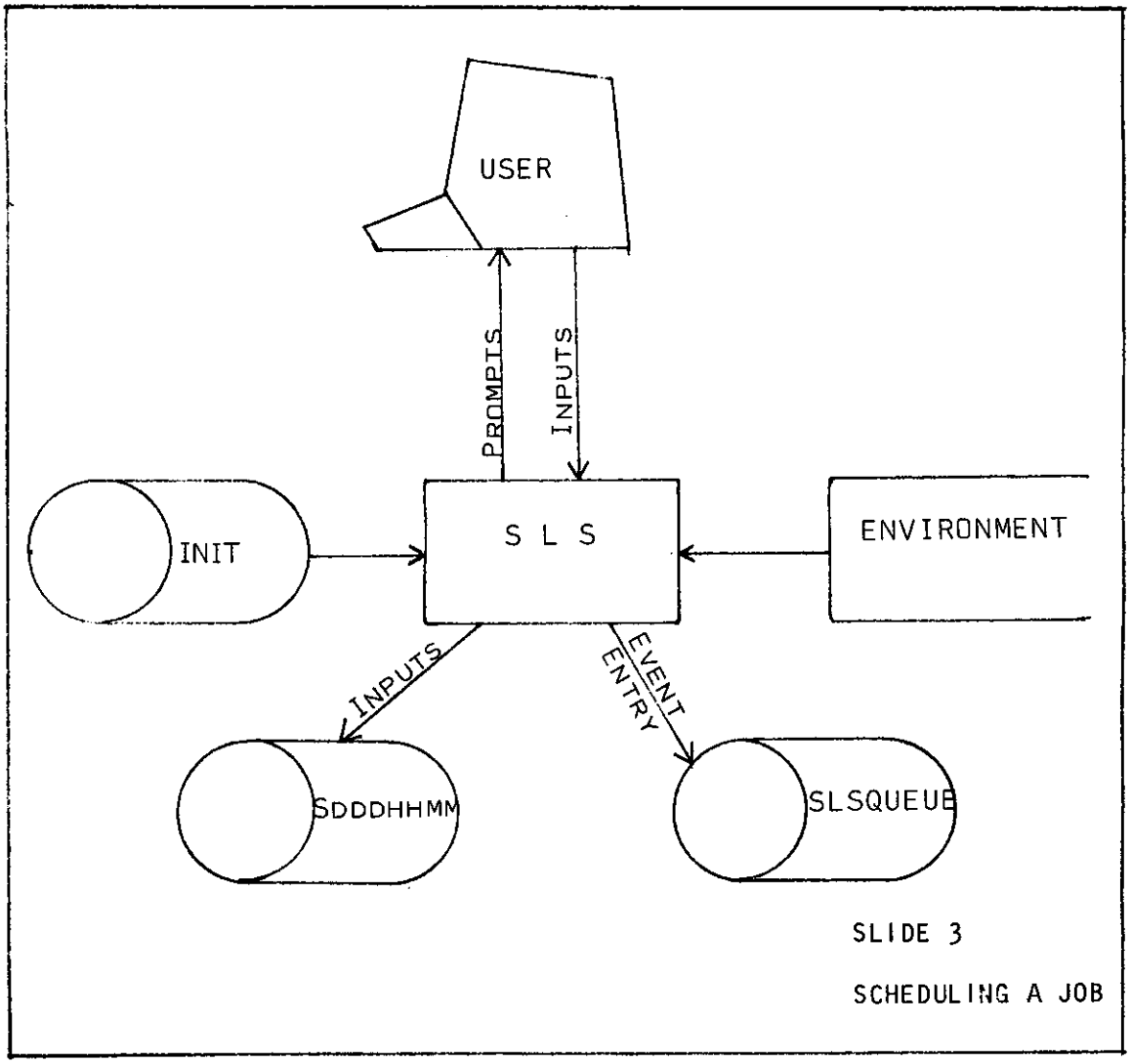
#J59 SLIDE 1

I would like you to note at this point some features of SLS:

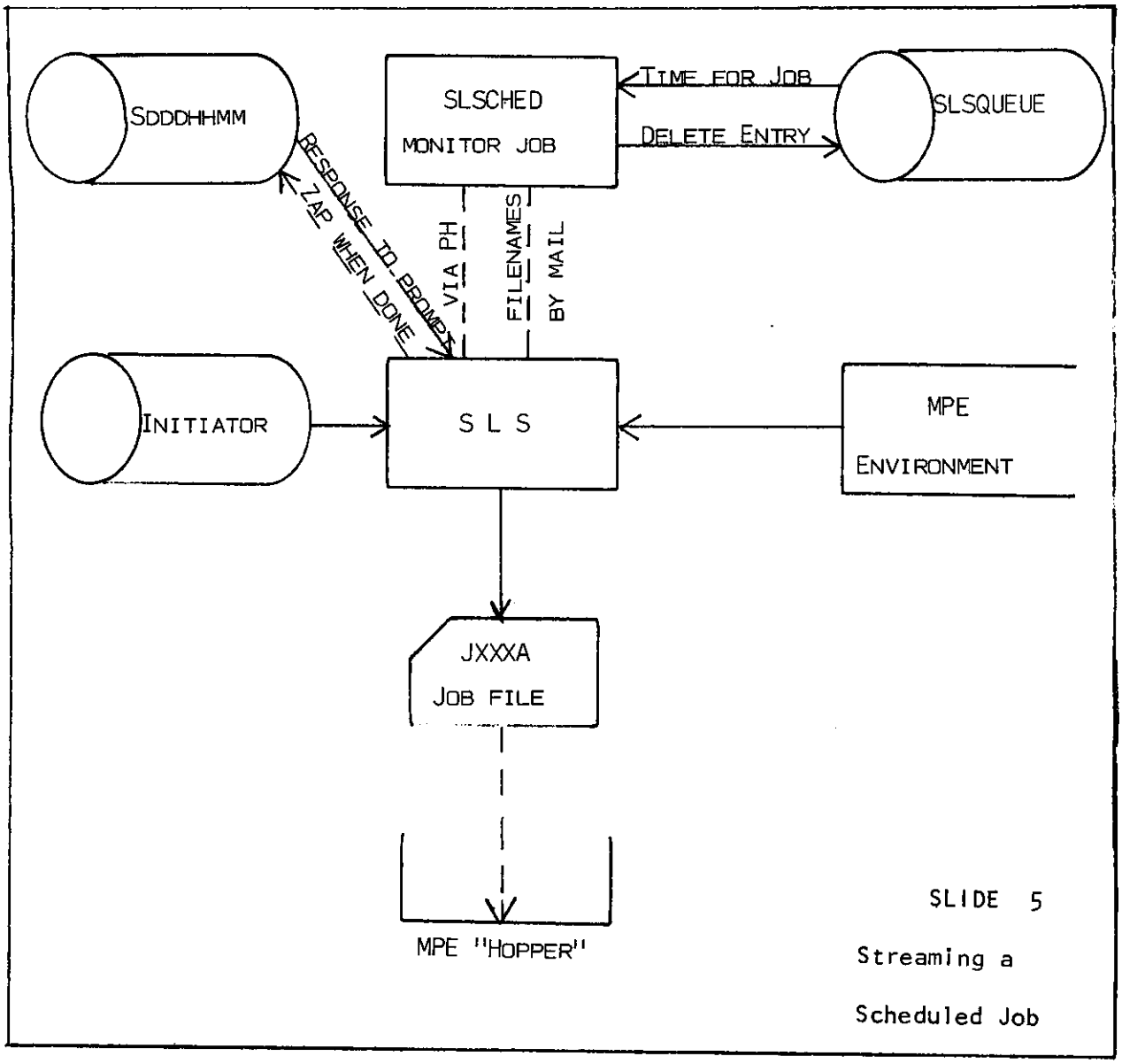
1. It produces a batch job, so the work will not detract from system response.
2. It obtained several variables from me. In one case my answer was not appropriate. SLS handled the problem on-the-spot.
3. When it is done, SLS will have produced for me the documentation I wanted. I didn't have to remember to ask--SLS asked me.



In the mode illustrated by the installation procedure, SLS operates under control of an initiator (template) file. SLS may obtain additional information from the computer environment or from the user. The final result is a configured job stream in a temporary file which SLS then streams.



```
:RUN SLSCHED.SLS.SYS


SLS SCHEDULEING QUEUE UTILITY
>S @.ADMIN
JOB  DATE      TIME USER              INITIATOR                  INPUT FILE
122  03/05/81  2310 ADMREC.ADMIN     ROSSLSI.PUB.ADMIN          S0641017.ADMREC
127  03/05/81  2310 ADMREC.ADMIN     ROSSLSI.PUB.ADMIN          S0641119.ADMREC
106  03/06/81  0100 FINPROG.ADMIN    DAYJOBI.FINANCE.ADMIN
111  03/08/81  0330 A900.ADMIN       UPDATEI.A900.ADMIN         S0071710.A900
18   03/09/81  0130 FINPROG.ADMIN    AWLETERI.FINANCE.ADMIN
37   03/10/81  0130 FINPROG.ADMIN    VOUCHI.FINANCE.ADMIN
75   03/11/81  0300 DWOMEN.ADMIN     PHONDIRI.DWOMEN.ADMIN      S2451623.DWOMEN
77   03/11/81  0330 DWOMEN.ADMIN     ROOMCHKI.DWOMEN.ADMIN      S2401458.DWOMEN
108  03/12/81  0130 FINPROG.ADMIN    INAUDITI.FINANCE.ADMIN
20   03/15/81  0030 CAFSYS.ADMIN     MIDMONI.CAFSYS.ADMIN
25   03/15/81  0040 STAFFPAY.ADMIN   INDEXI.STAFFPAY.ADMIN
26   03/15/81  2200 FINPROG.ADMIN    SELMO18I.FINANCE.ADMIN
27   03/25/81  0400 PAYROLL.ADMIN    STUWORKI.PAYROLL.ADMIN
29   04/01/81  0015 FINPROG.ADMIN    SELMOO8I.FINANCE.ADMIN
30   04/01/81  0030 CAFSYS.ADMIN     ENDMONI.CAFSYS.ADMIN
31   04/01/81  0040 CSHOP.ADMIN      CSRECAPI.CSHOP.ADMIN
>EXIT
END OF PROGRAM                                          SLIDE 4
:
```



In an alternative mode, the job file is discarded. Instead, a permanent disc file with all terminal input is saved. An entry regarding the date and time the job is to run is made in a file named SLSQUEUE.SLS.SYS. At the proper time, a monitor job will find the entry, run SLS against the initiator specified using the input file

saved for parameters, and stream the job.

Before we get any deeper into technical details, let's look at the seven functions of SLS:

1. Display of information on the user's terminal. The usual purpose is prompting. Any information to which SLS has access may be displayed.
2. Interrogation of user at the terminal, for information to be used in constructing the job.
3. Interrogation of disc files for information needed in constructing the job. An excellent application for this function is passwords. Passwords used in an application need only be kept in one place, with SLS looking them up as needed.
4. Interrogation of the operating environment. You can check for the existence of files, check sizes and empty space in them, and verify that there is sufficient room in a file for the data you propose to put in it. You also have access to many items in the WHO intrinsic. Optionally (uses priveleged mode), you may have access to the job/session ID of the perpretrator of the job.
5. Manipulation of information gathered in the steps above. This can include validation, translation, arithmetic, and many other functions normally associated with programming languages.
6. Preparing the actual job stream file. If we are merely putting it into the SLSQUEUE file, this product will be

discarded.

7. Submitting the job either:

   A. Directly, through the programmatic :STREAM command.

   B. Deferred, by adding to the SLSQUEUE file an entry specifying the initiator, date/time, and (if any input was requested of the operator of the terminal) parameters to be input.

These functions are performed under control of the initiator file, which is very similar to a computer program. SLS functions much like a language interpreter.

Now I'd like to show you how to use SLS. I'll use the quickest way I know--examples. Let's trace a universal application, system backup, through the various stages from where most sites are at to a fully SLS'd job that requires no operator intervention other than mounting tapes (and authorizing them). First, an HP-styled job stream:

```
BACKUPJ.MGR.SYS
   !JOB BACKUP,MANAGER/ALPHA.SYS
   !FILE TAPE;DEV=TAPE
   !SYSDUMP *TAPE
   NO                 << CHANGES >>
   0                  << DATE    >>
   !EOJ
```

SLIDE 6

The first thing we need to do is get the password out of this job. So we will create an additional file (IPASS.MGR.SYS) with the password to MANAGER.SYS on the first line:

```
IPASS.MGR.SYS
   ALPHA

                                    SLIDE 7
```

Now we will rewrite the job stream so it will get the password from this file:

```
BACKUPI.MGR.SYS
   << SYSTEM BACKUP INITIATOR VERSION 0.0 6/3/80 >>
   FILE IPASS.MGR MGRPASS 1 1 8
   PRINT !JOB BACKUP,MANAGER/<MGRPASS>.SYS
   PRINT !FILE TAPE;DEV=TAPE
   PRINT !SYSDUMP *TAPE
   PRINT NO                  << CHANGES >>
   PRINT 0                   << DATE    >>
   PRINT !EOJ                          SLIDE 8
```

Note the following differences:

1. The file name ends with ! instead of J.
2. Commands to be submitted are preceded with PRINT.
3. The variable MGRPASS is assigned a value by getting it from the file IPASS, and is referenced by <MGRPASS>.
4. You can give a copy of the initiator to anybody without fear, because they don't have your password.
5. Changing all jobs run by MANAGER.SYS for a new password can be accomplished by changing only the one password file.

Of course, it may be that you don't always want to do a full backup. The next version prompts the operator for the date to use:

```
BACKUP.MGR.SYS
   << SYSTEM BACKUP INITIATOR VERSION 1.0 6/3/80 >>
   << 0.0 06/03/80 INITIAL VERSION                >>
   << 1.0 06/03/80 ADDED "DATE" PROMPT            >>
   FILE IPASS.MGR MGRPASS 1 1 8
   3 INPUT DATE ENTER BACKUP DATE (DEFAULT: FULL BACKUP)?
   NULL 1
   YES 1 SET DATE 0
   YES 1 DISPLAY FULL BACKUP HAS BEEN SELECTED
   YES 1 GOTO 5
   DATECHK MM/DD/YY
   NO DISPLAY  INVALID DATE. ENTER IN FORMAT MM/DD/YY OR
   NO DISPLAY   CARRIAGE RETURN ONLY TO THIS PROMPT.
   NO GOTO 3
   5 PRINT !JOB BACKUP,MANAGER/<MGRPASS>.SYS
   PRINT !FILE TAPE;DEV=TAPE
   PRINT !SYSDUMP *TAPE
   PRINT NO                  << CHANGES >>
   PRINT <DATE>              << DATE    >>
   PRINT !EOJ
                                                SLIDE 9
```

Example of use:

```
:RUN SLS.SLS.SYS
JOB NAME?BACKUP.MGR
ENTER BACKUP DATE (DEFAULT: FULL BACKUP)?05/21/80
STREAM JXXXA
   #J28
END OF PROGRAM
:
                                                SLIDE 10
```

A nice "wrinkle" might be to add automatic selection of date in the case of relative backups. This is easy to do, using TODAY to get the date and FCOPY to put it on a file for subsequent retrieval by the FILE statement:

```
BACKUPI.MGR.SYS

   << SYSTEM BACKUP INITIATOR VERSION 2.0 6/3/80 >>
   << HISTORY:                                   >>
   <<  0.0 06/03/80 INITIAL VERSION              >>
   <<  1.0 06.03.80 ADDED "DATE" PROMPT          >>
   <<  2.0 06/03/80 AUTOMATIC DATE FOR RELATIVE  >>

   << FLAG USEAGE:
   <<  1  RELATIVE BACKUP                        >>
   <<  5  FULL BACKUP                            >>
   FILE IPASS.MGR MGRPASS 1 1 8
   INPUT DATE ENTER BACKUP TYPE: FULL OR REL?
   NULL 5
   YES 5 SET DATE FULL
   MATCH 5 FULL
   MATCH REL
   NO 1 NO 5 DISPLAY I NEED DEFAULT, "FULL", OR "REL".
   NO 1 NO 5 REPEAT

   << SET UP SYSDUMP DATE IN DATE VARIABLE     >>
   NO 5 FILE BACKDATE.MGR DATE 1 1 8
   NO 5 DISPLAY I AM INITIATING BACKUP RELATIVE TO <DATE>.
   YES 5 SET DATE 0
```

```
<<  CONSTRUCT THE ACTUAL JOB >>
PRINT !JOB BACKUP,MANAGER/<MGRPASS>.SYS
PRINT !FILE TAPE;DEV=TAPE
PRINT !SYSDUMP *TAPE
PRINT NO              << CHANGES >>
PRINT <DATE>          << DATE    >>

<< WE ONLY NEED TO CHANGE THE DATE IN BACKDATE IF WE HAVE >>
<< ACCOMPLISHED A FULL BACKUP.  HENCE, WE WAIT UNTIL THE  >>
<< SYSDUMP IS FINISHED (IE, HASN'T ABORTED)               >>
YES 5 PRINT !PURGE BACKDATE.MGR
YES 5 PRINT !BUILD BACKDATE.MGR;REC=-80,2,F,ASCII;DISC=2
YES 5 PRINT !RUN FCOPY.PUB.SYS
YES 5 PRINT FROM=;TO=BACKDATE.MGR
TODAY THISDATE MM/DD/YY
YES 5 PRINT <THISDATE>
YES 5 PRINT //
PRINT PRINT ! EOJ                                  SLIDE 11
```

The file BACKDATE.MGR.SYS will be used as a repository of the last backup date.

Note that as initiators get longer, the need for documentation increases.

For the final version, we will schedule this job to run every night at 10 PM. It will do a full backup on Sunday night, and relative on others. For further interest, we'll skip Saturday nights.

BACKUPI.MGR.SYS

```
<< SYSTEM BACKUP INITIATOR VERSION 3.0 6/9/80 >>
<< HISTORY:                                   >>
<<  0.0 06/03/80 INITIAL VERSION              >>
<<  1.0 06/03/80 ADDED "DATE" PROMPT          >>
<<  2.0 06/03/80 AUTOMATIC DATE FOR RELATIVE  >>
<<  3.0 06/09/80 SCHEDULE NIGHTS EXCEPT SAT   >>

<< FIRST FIGURE OUT WHEN TO RUN NEXT >>
DATECNV RUNDATE MM/DD/YY ; ; ; 1D
DATECNV RUNDAY WWW; <RUNDATE>;MM/DD/YY
LOAD <RUNDAY>
MATCH 5 SUN
MATCH 1 SAT
<< WE WILL NEED TO ADJUST BY 1 DAY >>
YES 1 DATECNV RUNDATE MM/DD/YY; <RUNDATE>; MM/DD/YY; 1D
SUBMIT <RUNDATE> 22:00
FILE IPASS.MGR MGRPASS 1 1 8
NO 5 FILE BACKDATE.MGR DATE 1 1 8
YES 5 SET DATE 0
PRINT !JOB BACKUP,MANAGER/<MGRPASS>.SYS
PRINT !FILE TAPE;DEV=TAPE
PRINT !SYSDUMP *TAPE
PRINT NO                  << CHANGES >>
PRINT <DATE>              << DATE    >>
YES 5 PRINT !PURGE BACKDATE.MGR
YES 5 PRINT !BUILD BACKDATE.MGR;REC=-80,2,F,ASCII;DISC=2
YES 5 PRINT !RUN FCOPY.PUB.SYS
```

```
YES 5 PRINT FROM=;TO=BACKDATE.MGR
TODAY FULLDUMP MM/DD/YY
YES 5 PRINT <FULLDUMP>
YES 5 PRINT //
YES 5 PRINT EXIT
PRINT !EOJ
SCHED
NO STOP
```

SLIDE 12

The last two statements bear comment. You don't want the job to be streamed when you use SLS to put it into the scheduler queue. So you use the SCHED statement to check if this execution of SLS is resulting from a schedule entry in SLSQUE. If not, just STOP to avoid streaming the job.

There are other capabilities of SLS. It is quite possible, for instance, to use SLS in an environment where terminals are managed by process handling. We provide SPL procedures packaged for non-SPL programmers, which do all the interaction with SLS for you. You need only code up a simple module in your terminal-handling program, and you can henceforth implement new job initiators your client can use without recompiling your program or even taking their process structure down!

The scheduling mode of SLS can be used for processing "un-jobs" we call events. An event is simply a processing of an SLS initiator that does things via the COMMAND intrinsic, and never actually streams a job. Every hour, for instance, we have an SLS event that sends beeps to a list of users who have requested that they be put on the "bong" list. Another SLS event is used at the end of the month to produce month-end reports and clear out useage counters.

After over two years of use (only one year with scheduling), what have been our results with SLS? We think it has been very worthwhile.

1. No longer do we forget to run repetitive tasks. When the student labor clerk comes in to do her payroll on the 25th of the month, her worksheet is already printed. That job

never runs while people are using the computer.

2. Our users are weaned from hovering over terminals from which they have run critical executions of programs. They put them into the job stream, then go off and do something productive.

3. With the reliability that comes from SLS's ability to check parameters for reasonableness, our clients are more willing to defer tasks so that they run outside of peak hours. This has helped our mid-afternoon response time greatly.

4. Forty percent of CPU time used by our administrative users in the last six months was done while there was NOBODY doing operations at the computer center. That, friends, was a person and associated family we didn't have to feed. That was a $10 reduction in the per-student cost of attending our school this year. That is a measurable difference in our organization's competitive position.

5. SMC has many dedicated workers. Some of them come in at strange hours during peak periods. I once caught the head of one of our user departments coming in at 3:30 AM. In former times, they had to have a computer person help if they were to get much work done. Now, the college benefits directly from such incidents--SLS preserves their productivity whenever they come in to work. That spreads out the load on our computer, in turn improving response time.

The results of our Computer Service Users Committee have been likewise encouraging. This committee started working in the fall of

1980, and has already accomplished a number of tasks:

1. Developed a working understanding among responsible people in our user departments, of events which affect response time. We have seen substantial efforts in the user departments as a result, to help us maintain good response time.
2. Given the DP management invaluable help in assigning priorities to pending software projects. This has been a two-way street. In some cases we have asked their indulgence while we delay technically demanding projects in favor of some less important but achievable.
3. Reallocated certain resources such as terminals, based on changing needs over our yearly cycle.
4. Begun to lay the groundwork for guaranteeing to the administration that a second CPU if purchased, would be effectively utilized but not "used up" until its depreciation cycle is finished.
5. Guided a subcommittee in the development of a resource allocation plan for students.

Terminal Spooling

In the abstract I promised to describe our experiences with terminal spooling. This topic is closely related to SLS and distributed operations. The final step in giving control to our users is giving them each all the visible elements of a computer. By installing printers in our major client departments, we have completed the task. The key was finding affordable printers. This was done by converting existing printing terminals to spooled devices.

The only change necessary was in the MPE software. Although this feature was announced for the Bruno release of MPE, we have been using it since release 1906. Thereupon hangs a tale...

One day a programmer at American Management Systems was talking to an SE about the problem of getting spool files out on terminals. He asked for some help in using SPOOK to capture spoofles on a printing terminal. The SE thought that was a strange way to do it. Why not just spool the terminal? That was a very significant question. I know a few other people who have asked it, with no response from the MPE lab. This time the right person knew the right thing. The result was a program, contributed on the San Jose swap tape last year, called SPOOLTRM.

There was quite some shock in some of our user departments when we asked if we could fix printing terminals so they couldn't be used to log onto the computer. When they found out that the result was the ability to send output from any other terminal to it, with all the advantages of the spooler, they tried it. We did learn several lessons. As HP begins to support spooling of terminals in the Bruno release, you may wish to take these into account:

1. The payroll people LOVE the fact that they now know that nobody can snoop at their printouts--which never leave the office.
2. They LOVE even more, the convenience. Our computer center is in a building separated by a courtyard from most administrative offices. Now they don't have to go outside to get three lousy pages.

3. We wish there were a simple method of limiting spoofle size to a specific device. But clients can learn pretty fast when their precious office printer is once tied up all afternoon, to route things properly. Again, SLS helps with this by reminding them.

4. Don't let it trouble your conscience if you tell them that a spooled terminal can only run one kind of paper. They will think they are really smart when they get it to do otherwise, and you won't have to put up with complaints when they blow it. But PLEASE don't expect most office workers to be able to understand special forms procedures in the spooler.

5. There is a price to pay--CPU cycles. It takes more of the CPU's time to service a request for another character on a terminal, than it does for that same character on a parallel-interfaced line printer (ie, 2613-2617-2619-2608). This factor is a particular problem for Series II and III users. In these systems, the CPU overhead for character transmission is higher because the ATC is a "dumber" interface than the ADCC used in the series 30/33/44. In a Series III running four spooled terminals at 2400 baud, this could result in a 15% overhead factor for handling terminal interrupts alone. To this one must add MPE overhead, swapping, and disc I/O.

Conclusions

Recently, a department which has the idea it is "terminal-poor"

had to give up one of its terminals for a few months. Their first reaction was to unspool the printer. After a staff meeting, however, they elected instead to make one of their CRT's mobile and retain the spooled state of the printer. They would rather share a CRT and occasionally have to go across the office for a lookup or update, than have to run to the computer center for every little printout. The day of the hard copy is by no means over.

Southern Missionary College, though it has only one CPU, has chosen to distribute control of the system to its users. We have done so because we feel there are advantages inherent in such a system. The DP management "control" functions are limited to executing procedures and policies set up by users for their mutual benefit, and packaging the system so that it is understandable to users. We feel this is the best way to get the most for our DP dollar.

We do not know what implications our experience has for systems with distributed intelligence. We see several factors that might encourage us to distribute our processing to multiple CPU's:

1. Physical lack of proximity. This is not currently a problem for us. If, for instance, we were to expand to our Orlando campus, this could be a significant factor.
2. Security. This is currently a prominent driving force towards acquisition of a second CPU. However efficient MPE may be in segregating users, it is impossible to assure our users that mere software can isolate applications as well as an electrical insulator such as two CPU's not interconnected would provide.

3. Dissimilarity of resource needs. However much we may prefer the HP 3000 for our administrative data processing, we are not convinced that it is necessarily the best system for academic processing. The 32,132 limit on adressing is a severe limitation compared to other systems of comparable cost. If the HP 3000 did not come with IMAGE, it would very possibly not be in the running.
4. Dissimilarity of useage patterns. Due to the way system resources are handled in the HP 3000, mixtures of dissimilar use can create severe response time degradation if certain users do certain things (like getting stuck in program loops). We feel that users who insure that they will not do such things, should not be subject to delays because others don't.
5. The HP 3000 can only handle so much of certain things, period. Examples include: directory limitations and excessive overhead in process creation. If you want over 6,000 sectors in your directory, you have to buy another system.

As things now stand, we feel that the HP 3000 as we are currently using it is an effective way of using the financial resources available to us for Data Processing. We have further discovered that a moderate amount of programming (creating SLS) and management effort (the Computer Service Users Committee) have resulted in multiplying its usefulness to our organization.