HEWLETT
PACKARD

CUSTOMER TRAINING

Helpful reference publications from HP

HP Documentation Index

publication #5953-2460

Lists more than 3,000 HP computer reference manuals.

Computer Users Catalog

publication #5953-2450

Describes HP operating supplies, magnetic media, cables, furniture, add-on hardware, and software for HP personal computers.

For your free catalog and/or index, telephone our Direct Order lines:

United States	800-538-8787	**West Germany**	07031-667829
California	408-738-4133		07031-223133
United Kingdom	0734-792868	**The Netherlands**	020-470639
	0734-792959		

Or contact your local HP Sales Office and ask for "supplies order processing."

HEWLETT PACKARD

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

G.H. DREHER
9/18/84

HP Computer Systems Training Course

SYSTEM PERFORMANCE TRAINING COURSE

Student Workbook

HEWLETT
PACKARD

#24 Inverness Place East, Englewood, Colorado 80112

Part no. 22809X

Printed in USA August 1983

COURSE CONTENTS

DAY 1
- Course introduction. An intro to OPT and general performance considerations.

- The contents of main memory. Basic segment structures and usages.

- Processes and the tables MPE uses to keep track of them.

- The LOADER and process creation and death.

DAY 2
- Memory manager (MAM).

- Schedulers and the MPE Dispatcher.

DAY 3
- The IO and File system. Disc characteristics, system architectures

- OPT logging and summary reports.

DAY 4
- IMAGE and file bottlenecks

- APS/3000 (SAMPLER).

DAY 5
- Other sources for performance data.

- How to make a sucessful performance analysis.

Labs will be done thruout the class and will be one of two types.

1) A "get to know it" lab where you exercise the tools under discussion against a known load.

2) A "super lab" where the system is placed under an unknown load and you use the tools to discover performance problems and remidies. (Super labs can be reconfigured by the instructor to meet local availability of resources and class needs. These labs may not be included in this workbook).

DEFINITIONS

SYSTEM: noun, A group of units so combined as to form a whole and operate in unison

PERFORMANACE: noun, To act, or process of performing

PERFORM: verb, Fulfil, to carry out, accomplish, to do in a set manner

(From the Merriam/Webster Dictionary 1974. G. & C. Merriam Co.)

For the purposes of this class SYSTEM PERFORMANCE will mean the HP3000 fulfilling it's function in a set manner

QUESTIONS

* What is the function of the HP3000 ?

* Can the HP3000 meet 100% of any user's requirements ?

* What is the purpose of analyzing the performance of the HP3000 ?

* Why isn't the system already optimized ?

* Do you have to understand each application on the system in greaat detail to analyze it ?

DISCUSSION

* The function of the HP3000 is whatever the user desires but should not exceed the capabilities of the system.

  -I.E. It would not be reasonable to expect the HP3000 to perform comparably with the CRAY-1 or a vegetable chopper.

  - It would be reasonable to expect it to perform a general DP function for a small to medium sized business or scientific user.

* The HP3000 will probably not be able to meet 100% of any users needs. It will, however, be capable of being configured to do the best job possible.

* You analyze the HP3000 performance to make the system come as close as it can to meeting the user's expectations.

  The most common reason for performance analysis can best be summed up by generalizing a user's complaints on the system.

  "It takes too long for certain programs to complete. What can you do to make the system go faster?"

  "The response time at my terminal is too long. What can you do to shorten it ?"

  It would be nice to be able to respond, as one TSE jokingly did, "Just type CONTROL-A G O F A S T E R on the system console". but no such command is yet a part of MPE. Instead we must attempt to understand WHY the system takes as long as it does then recommend specific steps to optimize it for this environment.

* The system IS already optimized but it is optimized for a general type of load. Since there is no way to know exactly what tasks a user will demand from their system then the system can't come pre-optimized for each site. Note that performance analysis is to customize the system performance for an individual load.

* Thankfully you don't have to understand each user's applications in great detail in order to analyze system performance. It is sufficient to understand what demands the applications place on certain key system resources and the system's ability to supply these resources.

CLASS OBJECTIVES

Provide the student with a knowledge of the variables that impact system performance

Enable the student to effectively utilize the graphic displays and hard copy reports of OPT/3000

Criteria for evaluating performance:

Effectiveness in handling a specific application

Internal effeciency of the machine

Although performance is subjective in nature, it can be quantitatively assessed by technical activities. the results may be objectively evaluated.

USES OF PERFORMANCE INFORMATION

System Configuration

Scheduling and Operations Management

System Tuning

Planning Upgrades

Software Evaluation

Control

LEVELS OF PERFORMANCE ANALYSIS

1) Observation and "Gut Feel" based on understanding

2) Standard MPE commands plus Contributed Utilities

3) HP supported performance tools

4) Consulting

DISCUSSION

1) **Gut Feel** is the first level of performance analysis. It should not be overlooked, even if more sophisticated tools are available. Gut Feel means applying a little observation and understanding to the system. For instance:

- Watch and feel your disc drives. if they are doing a rhumba that threatens to walk them out of the computer room then it is a good bet that one of your system bottlenecks may be disc IO.

- Further observe the disc drives. Does one appear to be accessed a lot more than the others. (Watch for a few minutes since this condition may change). If it appears that one disc is working harder than the others try to intuit why. Is it the system disc (ldev 1). If so this may indicate system directory activity. It may be virtual memory access (compare the locations of virtual memory with the disc being accessed). It may be one hard hit file or data set. Try to find out.

- If you have a CPU activity indicator (as on the series 30/33) or a Current Instruction Register (as on the 1,2,3, and 64) then see if the CPU has a significant amount of PAUSE time. The ACTIVITY indicator on the series 40/44 does NOT indicate pure CPU activity but includes IO also so it can't be used for this test.

- If the CPU is paused a lot and no disc drives are busy then either the user is not stressing the system or another type of bottleneck might be present (file locking, data base control blocks etc.).

I bet you didn't know that you could tell so much about a system by a simple "laying on of the hands" did you? Next time don't scoff at those old timers that walk up to a system, lean up against the system disc for a minute then suggest you buy more memory. They might be right!

2) **MPE** has several commands that may help you to get a first look at the system loading and response. Careful analysis of the SHOWJOB and SHOWQ listings can yield a lot of useful information. Running DBUTIL and doing a SHOW database LOCKS can help identify image locking bottlenecks.

Several very useful tools have been written and contributed to the **International User's Group Library**. Some of the contributed tools use Priv Mode and so should be used cautiously, especially when updating to a new release of MPE. Many good tools do not require Priv Mode and should not be dangerous. Analyzing the MPE log files, for instance, can provide much performance information without resorting to Priv Mode.

3) HP supported tools such as **OPT** and **APS/3000** (Sampler) can provide in depth performance data. They do use Priv Mode but, being HP supported, should not cause any problems on your system. They also have special "hooks" into MPE so that they can collect their data without themselves causing a performance problem.

4) Lastly, if you don't want to purchase one of the HP supported tools, if you need help interpreting them, or if the problems on your system are so complex that expert help is needed, then HP offers a **performance trained SE** to assist you. These SEs are kept up to date on all the new problems and features. They will also have access to some more complex HP tools that are not for sale to customers. (Usually because they require in depth training on MPE internals to properly run and interpret).

FACTORS AFFECTING PERFORMANCE

SYSTEM LOADING
(What should you measure)

1) Live Production

2) Simulation

3) Benchmark

DISCUSSION

There are several tools available for monitoring the system but you must provide the right workload to properly measure and optimize the system performance.

Live production is the best situation to monitor since that is actually what you will experience. It is not always possible to test under live conditions since in many cases one is not available to you. (Take the case where you want to forcast what will happen if you increase your current workload or changed your hardware configuration).

In any case you should always attempt to monitor the system in as close to a real-life situation as possible. The use of system resources is extreamly dependent on the type and number (mix) of applications being run.

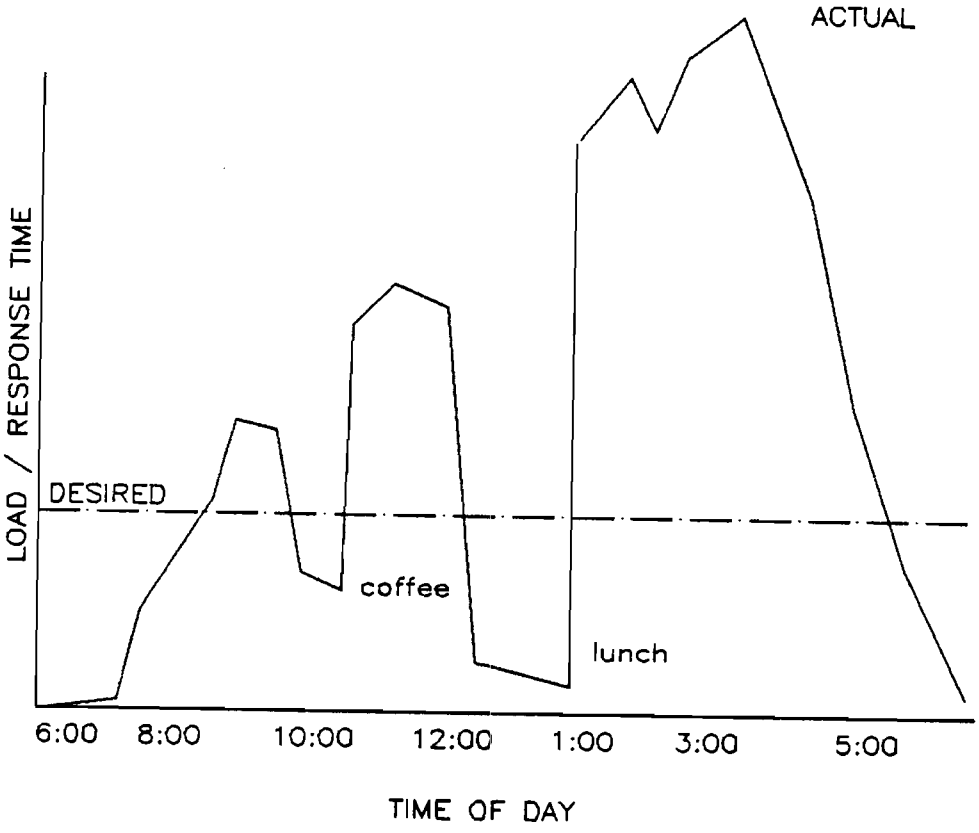
If it is impractical to monitor a real-life environment then you should arrange a simulation that comes as close as possible to it. This involves using the same amount of each system resource in approximately the same manner as you would expect in real life.

An artificial load (Benchmark etc.) that is not close to the real environment is almost useless. If you optimize the system using a benchmark then it will be good at that benchmark but probably even worse that before for on-line users. Note: Ignore this rule if the purpose of your system is to run benchmarks only.

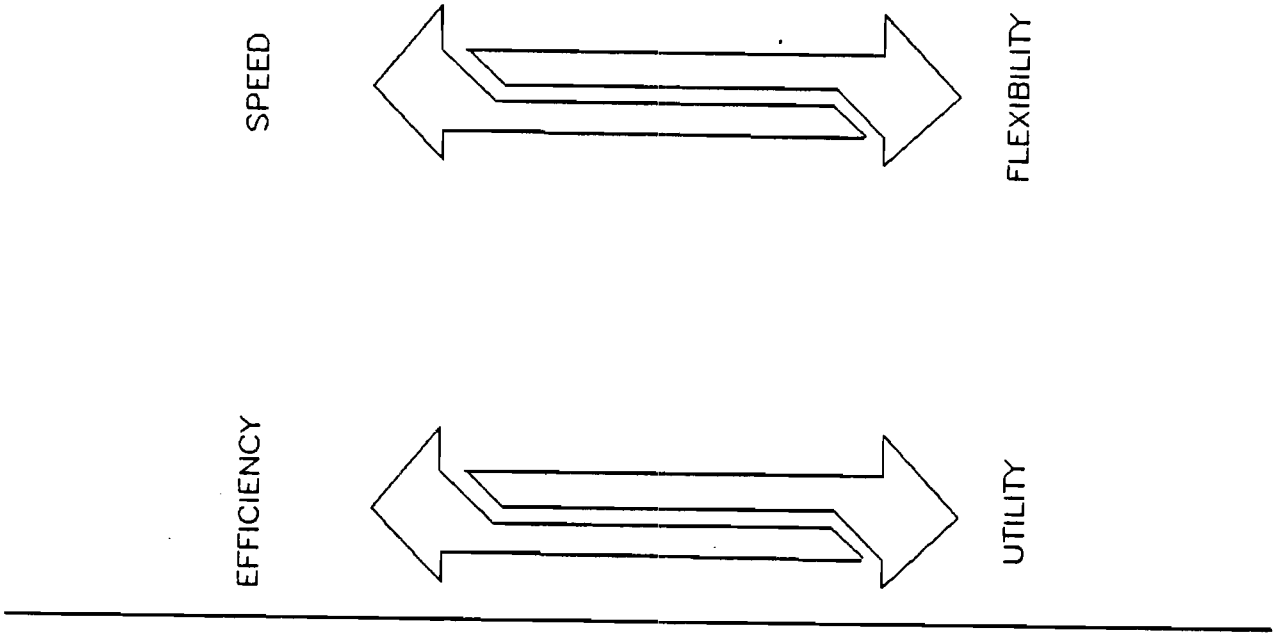
By the same token, it is also not best to monitor a system that is running the desired application but not under full load. It is difficult to forecast the interactions between programs or even between copies of the same program unless you actually run them. (How could you find a file locking problem if no one else ever locks the file ??). (One exception to this rule is where you already know the system's critical resources and want to modify an application to minimize its impact on them. In this case you can run a single copy of the program and monitor it with tools like APS/3000 [SAMPLER]).

In general optimizing system performance involves making tradeoffs between system resource usage. If you don't analyze the system under full load then you might end up with each application running well by itself while the system as a whole is still lousy.

(When do you measure)



TRADEOFFS



LOADING ALTERNATIVES

SCRIPTING

MODELING

TEPE

DISCUSSION

SCRIPTING is the process of modifying the programs under test so that they do not require operators to run them. This usually involves creating SCRIPT files which contain the user input then modifying the program to read these files instead of waiting for the user to type them in.

Points to be careful of when scripting: Be sure to include a call to the PAUSE intrinsic to simulate user think time. Don't simply do a "do nothing loop" since this chews up CPU time and will skew your results. Consider the MPE dispatcher. It adjusts a process priority each time a terminal read is done. If you are no longer doing reads from the terminal then this may skew the test results. (Some users have been able to get around this by calling READ and specifying a buffer length of zero).

Lastly, since this method involves modifying the programs under test it will bias the results somewhat. Take this under consideration when analyzing your results.

MODELING is the process of taking some raw data from your system and running it thru a modeling program. these programs would be most helpful when you need to test actual programs under a hardware or software configuration that was not available to you. A good example is the DISC CACHE MODELING program.

TEPE is by far the most reliable method for performance testing. It uses another system to enter the SCRIPTS directly into your system's terminal ports. Since the simulation is done on another system then it can not bias the system under test. Another benifit to TEPE is that, once the scripts have been prepared, a test can be repeated EXACTLY the same as before on different systems and configurations.

The drawbacks to TEPE are: Preparing the scripts can take a significant amount of time (1-3 weeks per application). To run TEPE you must have at least two systems, one to run TEPE and the other one to run the user applications. Lastly, TEPE is not currently available outside of the HP performance testing Lab.

RESOURCES AND BOTTLENECKS

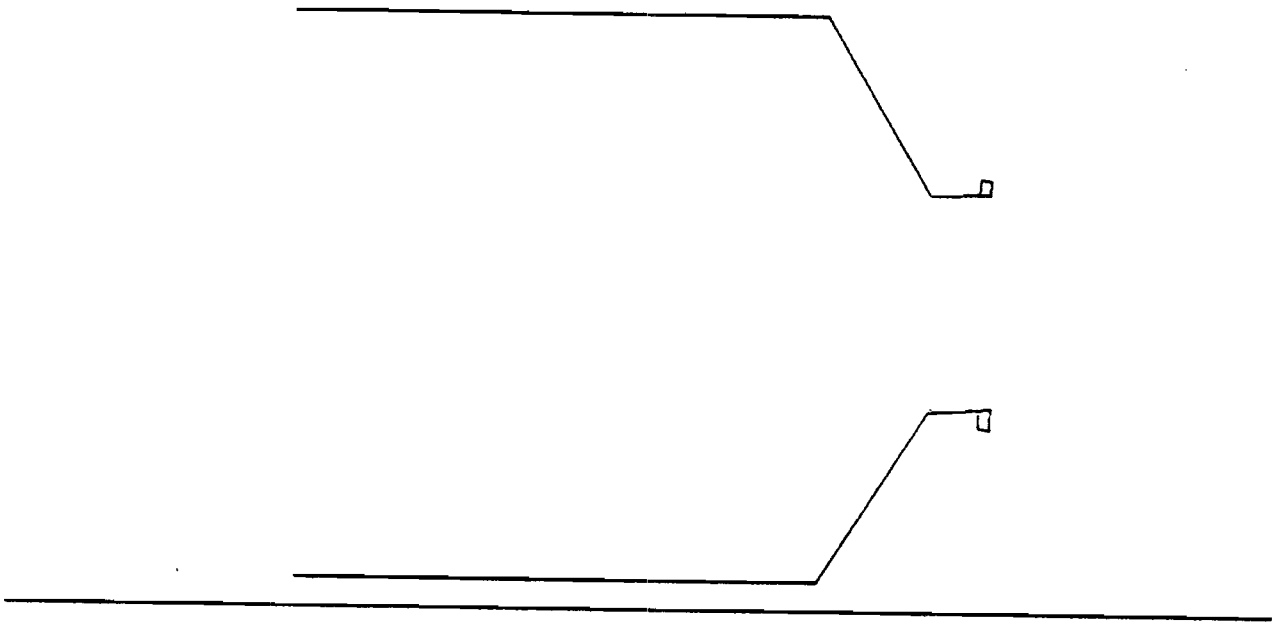
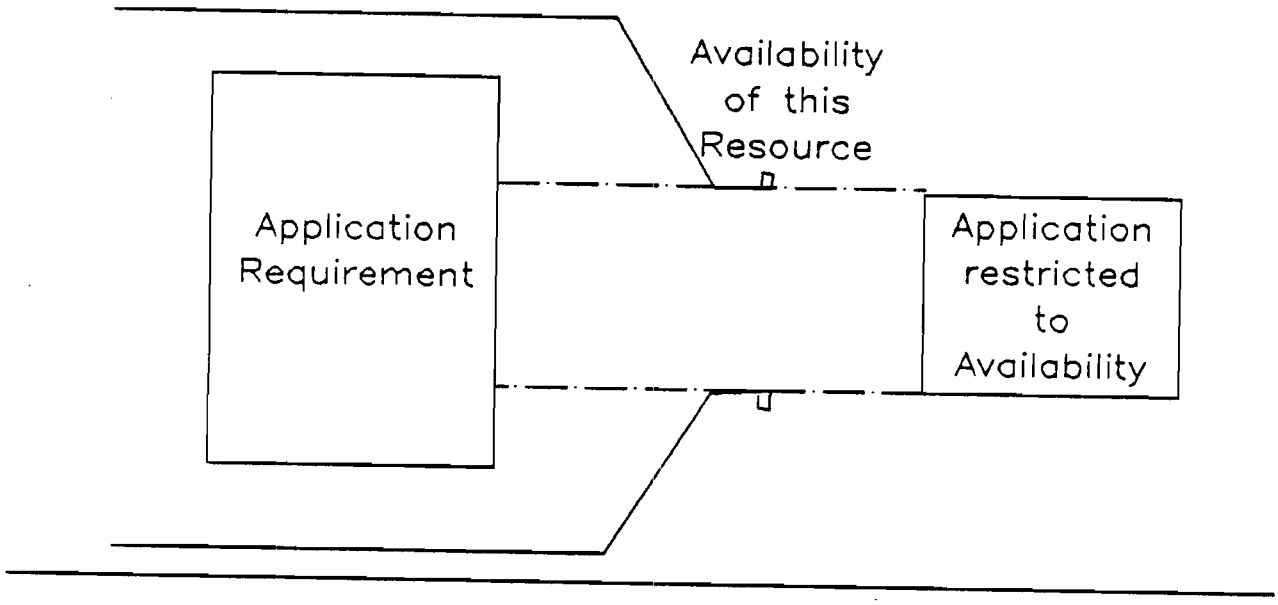


DIAGRAM OF A BOTTLENECK



Potential Bottlenecks

CPU Speed

Available Memory

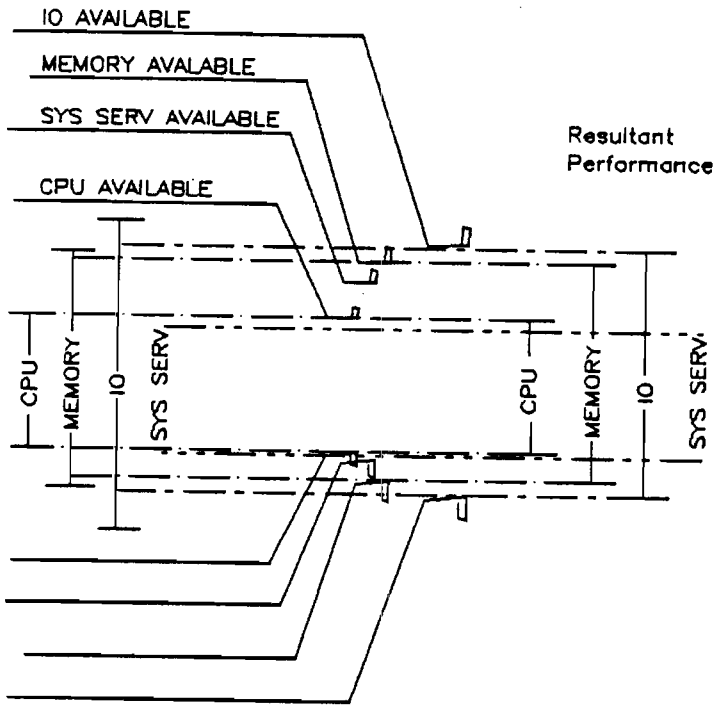
IO Thruput

System Services & Tables

All based upon the application's ability to demand a resource and the system's ability to deliver it

---

AN EXAMPLE



Note: Increasing the availability of a resource will only help performance if the applications are demanding more of that resource than is currently available

MURPHY'S RULE

OF BOTTLENECKS

There is ALWAYS one more Bottleneck !

TRADEOFFS

You can't get something for nothing
but very often you can make a trade

Trade off using more of a resource

you have in excess to relieve a resource

that is exhausted

FEATURES OF OPT/3000

INTERACTIVE REPORTING

GRAPHIC PRESENTATION

MULTIPLE LEVEL DISPLAYS

SIMPLE AND FLEXIBLE USER INTERFACE

HARD COPY CAPABILITY

PERIODIC SUMMARY REPORTING

ON-LINE DOCUMENTATION

LOG FILE

OPT INTRODUCTION LAB

This lab is to acquaint you with the On Line Performace Tool (OPT). Don't attempt to understand everything you will see as it will be discussed later. The main thing you should do is to get familiar with the command structure of OPT and be able to find your way around it.

Read the OPT Manual, sections 1 thru page 2.3

:RUN OPT.PUB.SYS

The first display you will see is called the GLOBAL display. It contains information on the primary system resources and their usage.

Enter the following commands. Note that these commands will NOT be echoed to your terminal. If OPT is updating the terminal display then it will ignore your commands. You should NOT have to press the RETURN key at the end of a command. Any invalid command will not be acted on but the terminal display will be updated instead.

The **H** command enters an on-line HELP facility. The information presented will vary depending on what OPT display you are currently looking at. HELP provides a lot of information on how to run and interpret the various OPT displays. Learn to use it often. Try it now.

The **?** command will produce a menu of available commands. Like HELP, this list of commands is tailored to the OPT screen currently being displayed. Try it.

Notice that some OPT commands are single characters while others require multiple characters.

Read the OPT manual, pages 3.2 - 3.16 for a description of the commands you can enter to control the displays.

On your terminal. Enter the **#?** command for a menu of the display control commands.

Now, using the first display (GLOBAL RESOURCE DISPLAY) answer the following questions. HINT, You can always get to the global resources display by typing **#G**.

1) How full is main memory ?

2) Is the CPU being kept busy all the time ? If not then how busy is it usually ?

3) How fast are the disc drives going ? Can they go any faster ?

4) What does the "P" mean in the CPU display ?

5) What can you tell if you see no "I" in the CPU display ? Are there any exceptions ?

Can you find the command used to terminate OPT. (There are actually two of them). When you are thru with this lab, exit OPT.

CHARACTERISTICS of the RESOURCES

CPU:

Very fast, expandable only by replacing the system

MEMORY:

Fast, expandable to a limit, extendable by swapping to disc (much slower)

DISC:

Extendable, speed determined by devices, and controller selections. Disc cache can extend speeds

TERMINALS:

Speed depends on controller, terminal and method of connection to system. (block mode vs character mode etc.) System performance related to the number of terminals.

DISC SPACE:

Set by types of disc drives. Available space and fragmentation can affect system performance.

SYSTEM TABLES:

Configurable by system manager, an excess can waste memory but a shortage can be disasterous. Some tables are memory resident and therefore more important to configure.

OTHER DEVICES:

Should be configured to match their intended usage (print load, backup of disc drives etc.)

SYSTEM TABLES LAB

Run OPT.

Enter the SYSTEM TABLES context.
(Remember how to find out what command to enter)?

What can you tell from these displays ?

Who would normally use them ?

What contributed utility programs produce similiar displays ?

MEMORY

WHAT IS IN MAIN MEMORY ?

CODE SEGMENTS

DATA STACKS

EXTRA DATA SEGMENTS

DISC DOMAINS

MPE TABLES

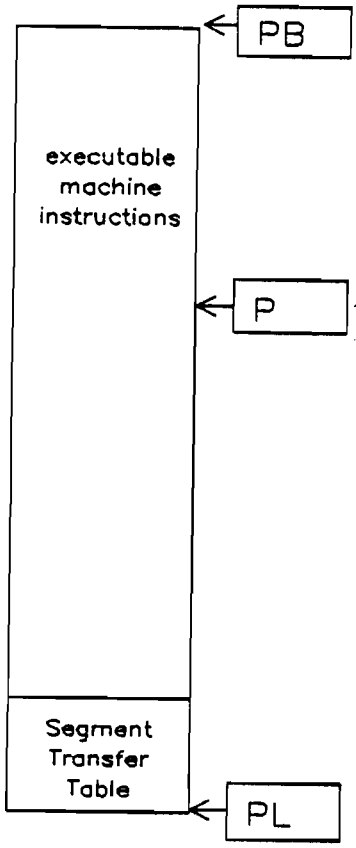
CODE SEGMENTS

Code segments contain executable machine instructions

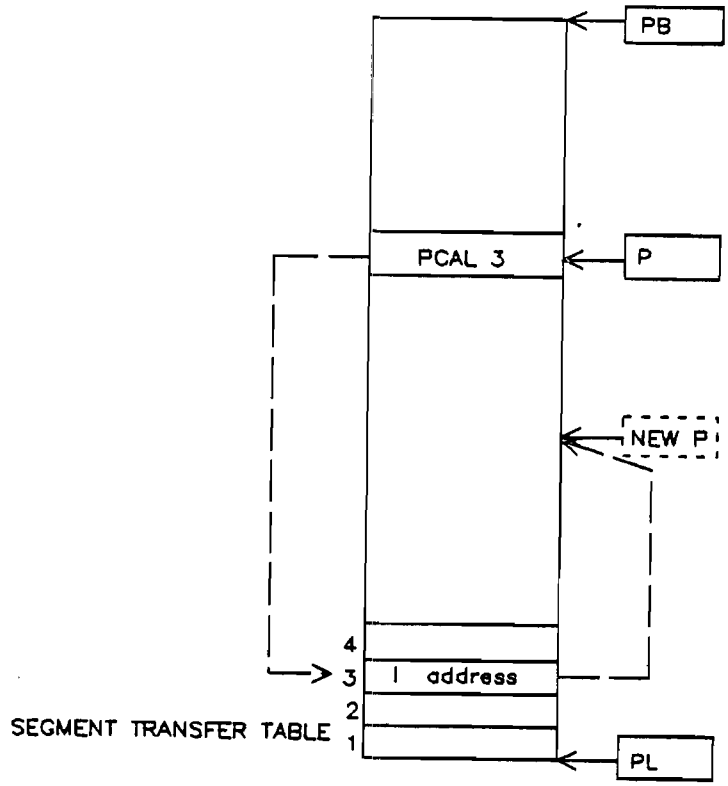
By definition, anything in a code segment can only be read, NEVER MODIFIED !

This gives the HP3000 REENTRANT (sharable) code.

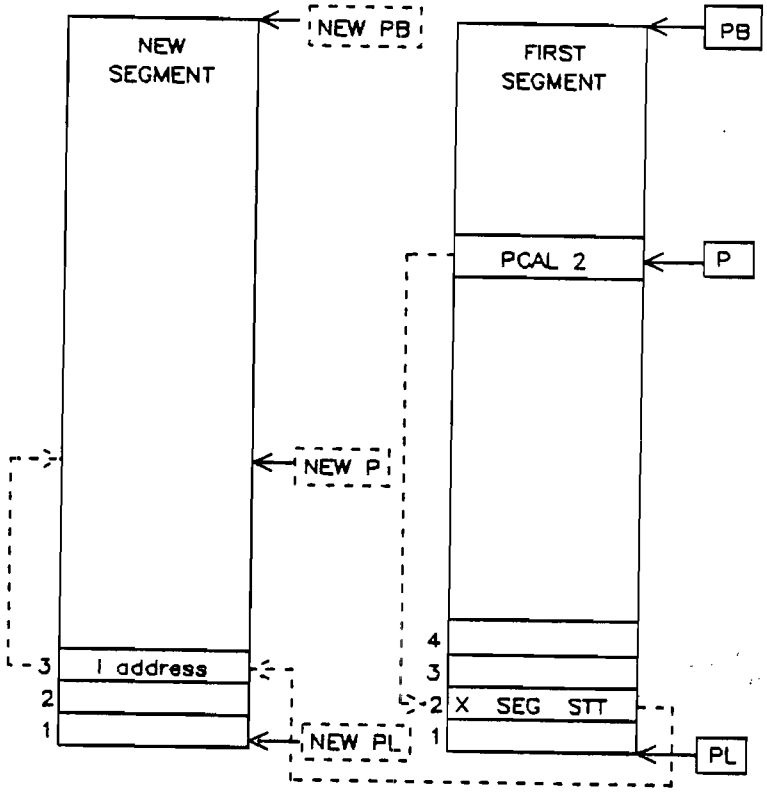
A process can have many code segments



CALLING AN INTERNAL PROCEDURE



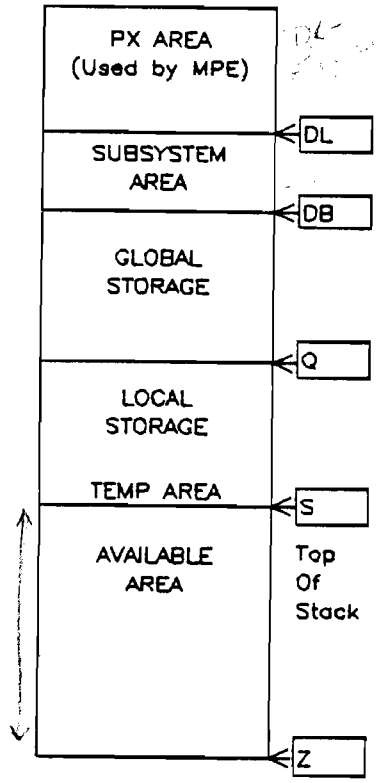
CALLING AN EXTERNAL PROCEDURE



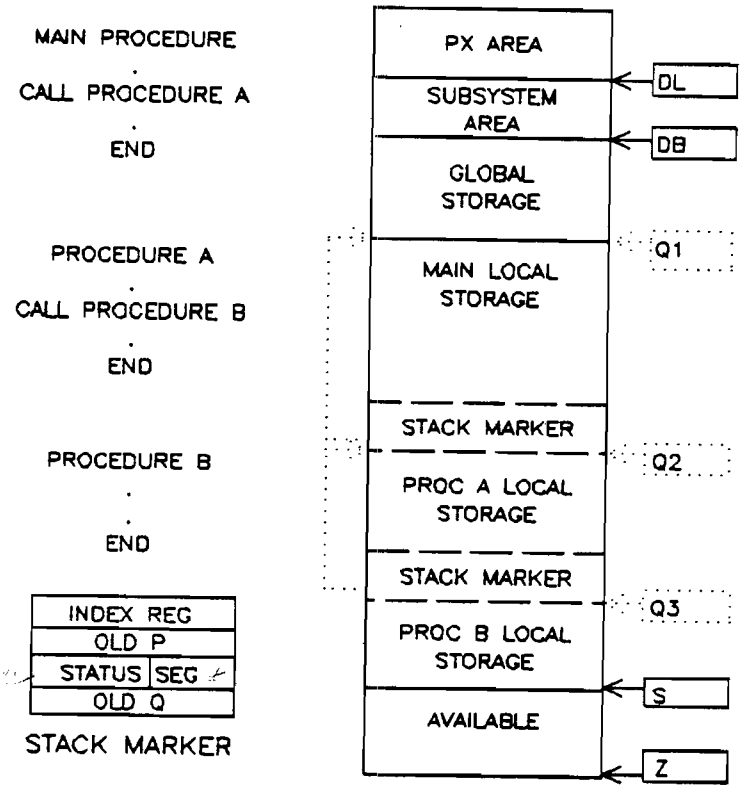
A DATA STACK

Data in the stack is modifiable and thus is NOT sharable

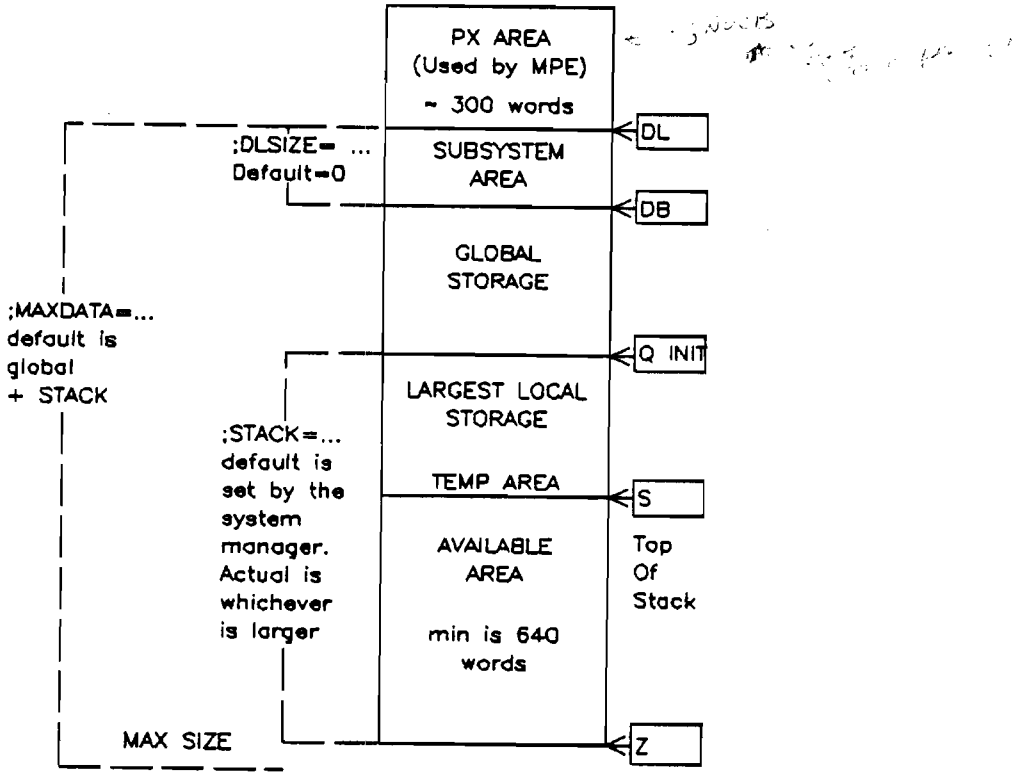
A process will have exactly ONE data stack



DYNAMIC STORAGE ALLOCATION



HOW BIG IS THE DATA STACK ?



LANGUAGE CONSIDERATIONS

SPL: Outer Block uses GLOBAL storage
Procedures use LOCAL storage
unless variables declared OWN

FORTRAN: COMMON and FLUT in GLOBAL
Main program and subroutines all use LOCAL

BASIC (Compiled): COM area goes into DLAREA
BASIC variables & pointers are GLOBAL
FILE buffers are GLOBAL
Program variables are LOCAL
(INVOKE works like PCAL)
(CHAIN reuses current LOCAL)

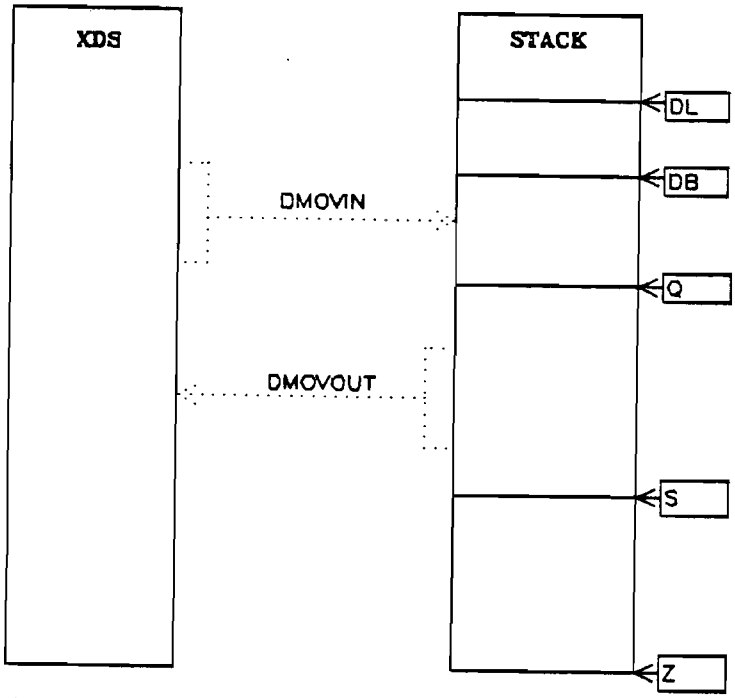
COBOL: All storage is GLOBAL except
Dynamic Subprograms which are LOCAL

PASCAL: The HEAP is in the DLAREA
Main program uses GLOBAL storage
Procedures use LOCAL storage

RPG: All storage is GLOBAL

EXTRA DATA SEGMENTS

Extra Data Segments (XDS) are unstructured storage.
No CPU registers point to them so data must be moved into the data stack to be used.
The data movement is entirely under program control



A process may have up to 255 Extra Data Segments

WHAT ARE EXTRA DATA SEGMENTS USED FOR ?

1) MPE file system buffers

2) KSAM buffers

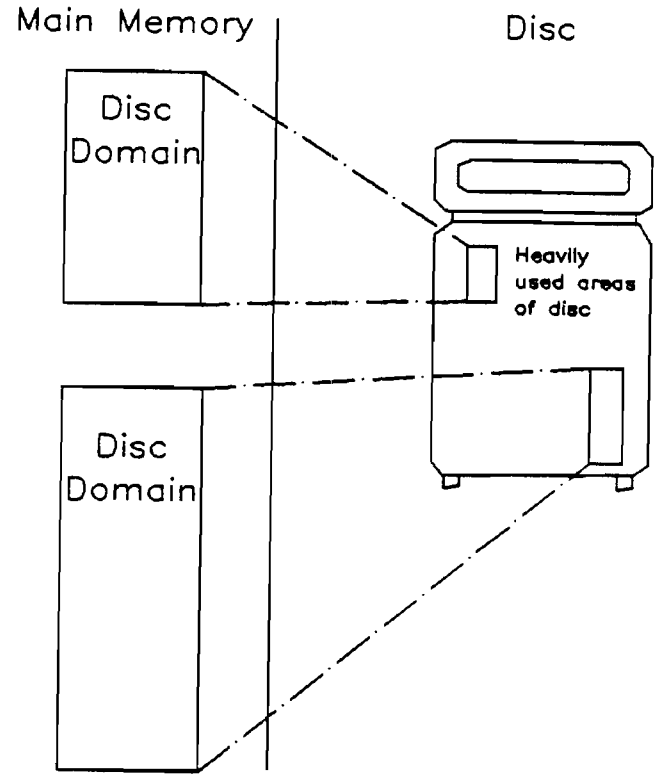
3) IMAGE data base control blocks & buffers

4) MPE tables

5) User programs can use them any way they want.

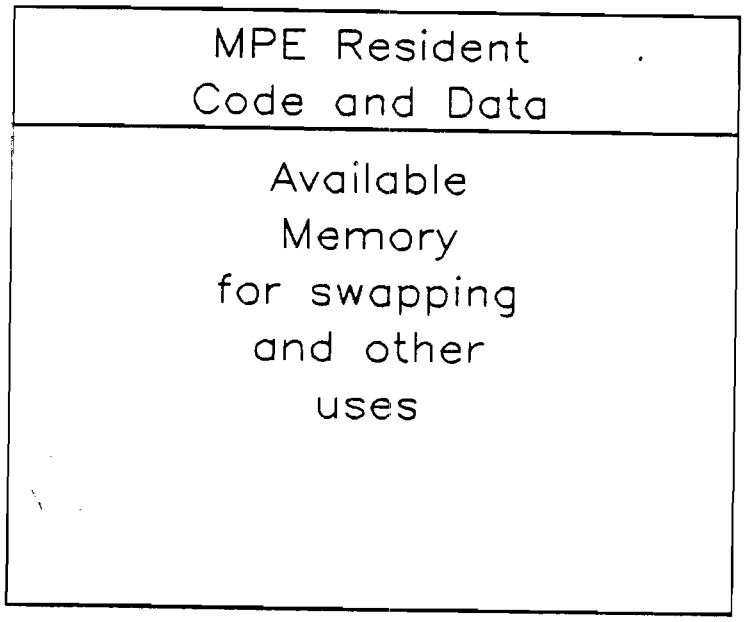
DISC DOMAINS

If disc caching is being used then you may also find disc domains in main memory

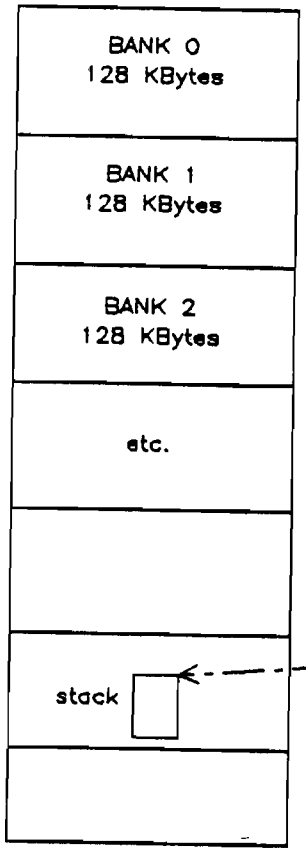


Memory Resident MPE

Certain parts of MPE must always be in main memory. They include critical and often accessed tables and programs. These are placed in memory whenever MPE is first loaded then marked so they can never be "SWAPPED OUT".



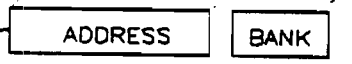
HARDWARE LAYOUT OF MEMORY



The HP3000 uses a 16 bit word to address memory

A 16 Bit number can address 65,536 locations. This is one BANK of memory

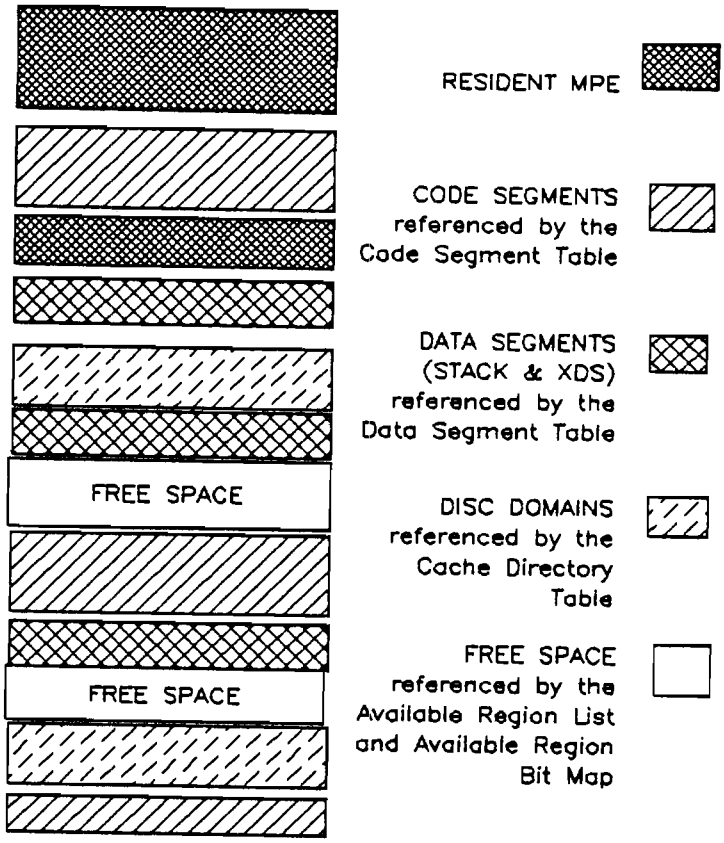
All user addressing uses 16 bit numbers plus one of the CPUs special purpose registers DB, Q, S, PB, P These registers have an extension to indicate what bank they are in.



DB REGISTER

For this reason, no segment can cross from one bank into another.

SOFTWARE LAYOUT OF MEMORY



MEMORY CONTENTS LAB

1) Log on and RUN OPT.PUB.SYS

2) Select the "Memory Report Display" **(#MR)**

3) How much memory does this system have ?

_____ Banks of 64Kwords each = _____ words.

4) What percentage of main memory is currently being used by:

_____ Resident MPE

_____ Code Segments

_____ Data Stacks

_____ Extra Data Segments

_____ Disc Domains

5) What is the average and maximum sizes of the following:

How many of each are in main memory and how many are not ? I.E. How many are swapped in and out ?

	AVERAGE	MAXIMUM	IN MEMORY	SWAPPED OUT
The Code Segments				
Data Stacks				
Extra Data Segments				
Free Space				

6) Take a look at the "Memory Contents Display". **(#MM)**

Where is Resident MPE in memory ?

Which bank has the most free space ?

What does it mean when an area is blinking ?

Can you find any IMAGE control blocks in memory ? How do you know which areas these are ?

Why is all the free space at the end of each bank ?

7) Use the "Bank Contents" display to examine a bank. What can you tell about the free space that you couldn't tell from the "Memory Contents Display"?

8) Why do some areas change colors ? What do the colors mean ?

PROCESS

WHAT IS A PROCESS ?

DEFINITION:
A Process is the unique execution of a particular program by a particular user at a single point in time.

EXAMPLES:
1 user runs 1 program once

1 user runs 2 programs once each

1 user runs 1 program twice, one after the other

1 user runs 1 program twice at the same time

2 users run 2 different programs at the same time

2 users run 2 programs at different times

2 users run the same program at the same time

1 user runs two copies of the same program at the same time. (difficult but possible)

How many processes are there in each of these cases ?

PROCESS COMPONENTS

Each process will have EXACTLY one Data Stack.
The stack is NEVER sharable between processes.

A process will have one or more Code Segments.

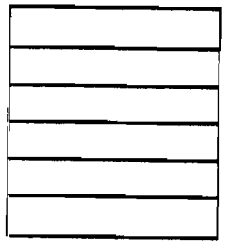
1-63 (Pre MPE-V) or 1-255 (MPE-V & Later)
code segments from the program file.
These segments are shared by any process running
this program file.

0-192 (Pre MPE-V) or 0-2048 (MPE-V & Later)
code segments from segmented libraries.
(Max of 255 per program on MPE V & Later)
These segments can be shared by any process
regardless of which program is being run.

Various and sundry entries in the MPE tables
used to locate and control the process.

Various MPE tables keep track of the pieces and status of each process.

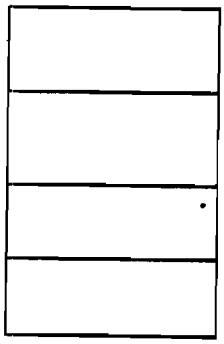
PCB



The Process Control Block (PCB) contains pointers to:

* The Data Stack
* The Code Segment currently being executed
* The Extra Data Segment currrently being accessed (if any)
* The Segment Locality List (SLL) in the Swap Table.
* The "Status" of the process
* Links to related processes

SWAP TABLE

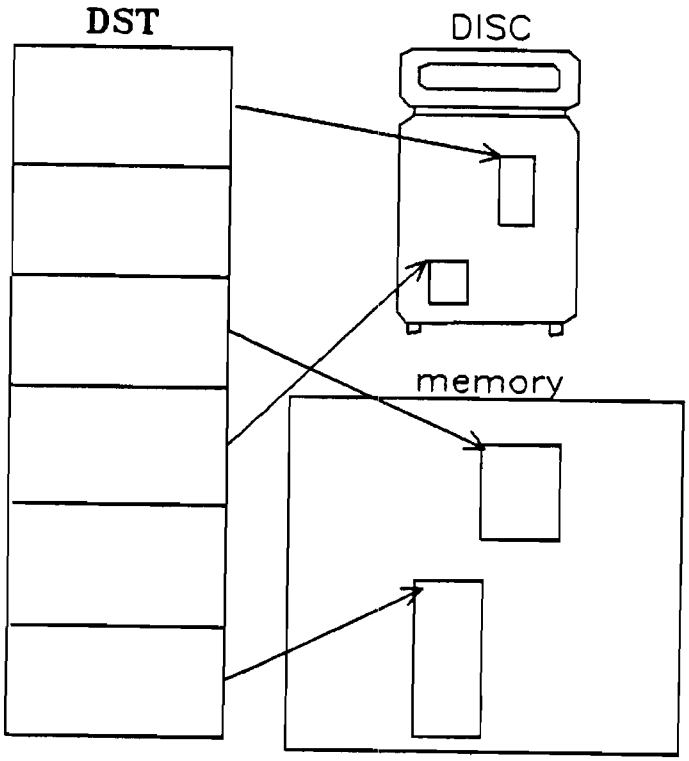


The Swap Table contains the SLL tables for each process. Each SLL keeps track of the segments (Program Code, SL Code, and Data) that this process has in memory.

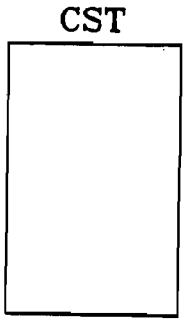
These segments make up the "WORKING SET" for the process and is maintained my the memory manager.

DATA SEGMENT TABLE

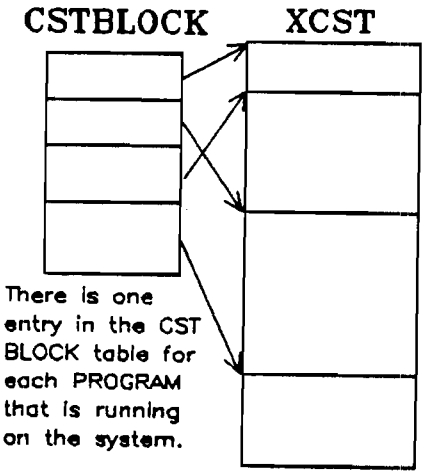
The DST contains pointers to all Data Segments (XDS & STACK) that are in memoy or on disc.



CODE SEGMENT TABLE
EXTENDED CODE SEGMENT TABLE



The CST points to all Segmented Library code segments that are candidates to be swapped into memory. These segments may be either in memory or on disc. (Same format as the DST)



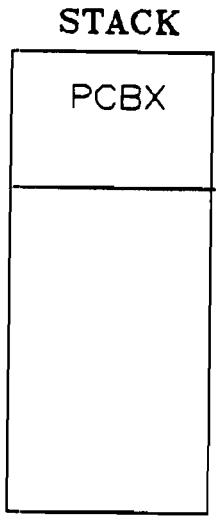
There is one entry in the CST BLOCK table for each PROGRAM that is running on the system.

The XCST contains blocks of CST entries, one for each program file that is being run on the system.

Each block points to the code segments for this program. One entry in the CSTBLOCK table points to the beginning of the XCST table entries for each program.

The format is the same as for the DST and CST.

PCB EXTENSION AREA (PCBX)



The PCBX is in the process' STACK. It contains informatin specific to this process such as:

* Process local XDS
* Trap procedures (Control-Y etc).
* Performance Statistics
* Open files & file buffers.

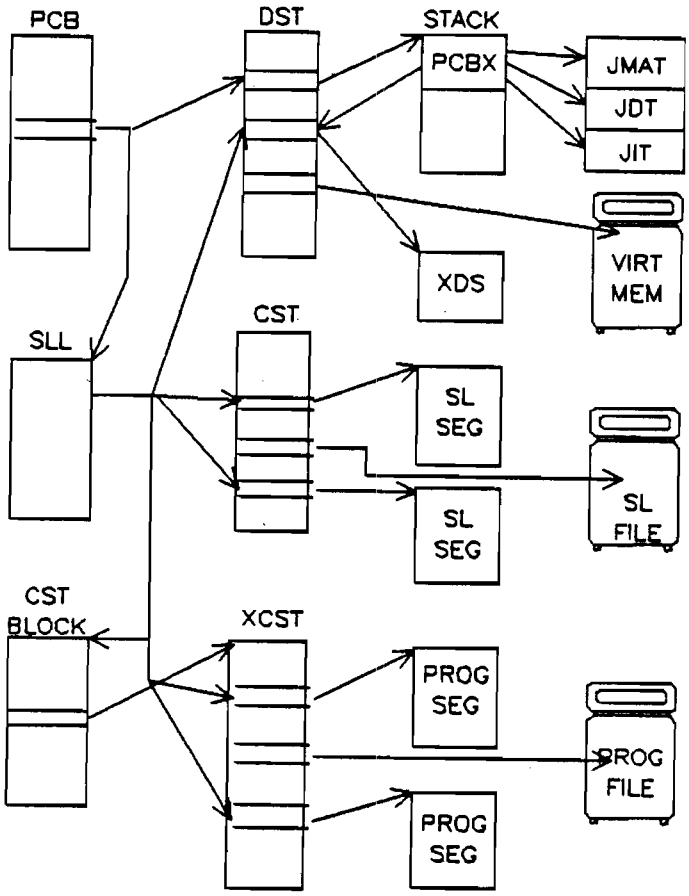
It also points to additional MPE tables for this process, such as:

*JMAT (Job Master Table)
User Name
CPU limit
Job/Session #
Execution priority

* Job Directory Table (JDT)
File equations
Temporary files
JCW's
Job Global XDSs

* Job Information Table (JIT)
User capabilities
RINS
Private Volumes

HOW THE PROCESS TABLES FIT TOGETHER



DISCUSSION

Where do the system XDSs go ?

At logon, JIT, JDT, 1 buffer for $STDIN/$STDLIST, CI STACK => 4 XDS

If UDC's are in effect then add 1 XDS for a UDC directory + 1 XDS for a file buffer for each UDC file.

:RUN program => 1 XDS for the process stack + 1 for each buffered file opened by the program.

If any files are on a private volume then add 1 XDS for the Private Volume Directory segment + 1 for each Private Volume User.

As you can see the use of XDSs can go up rapidly depending on what system features are being used. For example:

The minimum XDS for each user to LOGON and run a simple program is;

LOGON => 4 XDS :RUN => 1 STACK + average of 2 file buffers (or seven total dst entries).

50 users then require a minimum of 350 DST entries, not counting MPE.

If, on the other hand each user used 3 UDC files and accesses one private volume then;

LOGON => 4 XDS UDCS => 4 XDS PV => 1 XDS (+ one for the system) :RUN => 1 STACK + 5 file buffers (a little larger program) (or 15 DST entries)

50 users would now require 751 DST entries.

As you can see, the system can rapidly be limited by the size of the DST table. Solutions are to minimize the usage by each user. For instance, combining the UDC files into a single file can save 2 DSTs per user or 100 total.

Consider what happens to the above examples when you try to run 200 users on a series 68. How about 400 users ???

PROCESS LAB

1) RUN OPT.PUB.SYS
   Select the "User Summary Report" **(#P)**
   Identify the PIN, USER NAME, PROGRAM NAME, and WORKING SET INFO
   for your session.

2) Where does OPT find this information ?

3) What does it mean if a pin is highlighted (inverse video) ?

4) Does this display show all the processes running on the system ?

5) Go to the "Program File Display" **(F)**

   How does this display differ from the User Summary Report ?

   What does the asterisk preceeding a program name mean ?

   Does this display show all the programs on the system ?

   What does the 'I' command do ?

6) Examine a program with at least one process running it **(U)**.
   This command will take you into the "Program File User's Display".

   What additional info does this display give you ?

   Can you get even more information from here ? How ?

WADER

(The birth and death of a PROCESS)

THE TWO PHASES OF BIRTH

CREATE:

1) Reads the program file
(if program is already being run)

2) Getan entry in the CSTBLOCK table.
Get proper number of entries in XCST.

3) Load pointers to to program code segments into CSTX.

4) SearchSLs for unsatisfied externals.
loads pointers to SL segrnents into CST.

5) "Fixes up" STT entries in program code
to point to correct CST & CSTX numbers.

6) Obtoin space in virtual memory area for
STACK. Initialize this area.

7) Load pointer to stack into DST.

8) Get PCB entry and load pointers
to STACK, Initial Code Segment etc.
Mark status as NOT ALIVE.

ACTIVATE:

1) Mark process status as ALIVE

DEATH OF A PROCESS

TERMINATE, a system intrinsic, is called either explicitly or implicitly whenever a process terminates. MPE will call it if the process aborts.

What TERMINATE does:

1) Close any files the process has open.

2) Release any other local resources (RINs, local XDS, etc).

3) Mark the PCB entry as "dieing"so the process won be dispatched.

4) Remove the DST pointer to the STACK then release any virtual memory it held.

5) Remove the PCB entry for this process.

6) If no other processes are running this program file then;
   - Release the XCST and CST BLOCK table entries.
   - Release any CST entries that no other programs are using.

NOTE: It does take CPU time for a process to die.

:ALLOCATE

Q) What does "ALLOCATING" a prorgam really do ?

A) It makes it appear that the program is
being run even when it isn't.

ALLOCATING a program goes thru steps
1–5 of process creation.

It does NOT create a data stack or obtain
a PCB entry but it DOES use CST, XCST,
and CST BLOCK table entries.

Q) What does ALLOCATING a program buy you?
What does it cost you?
WHEN does it save and when does it cost ?

A) Class participation

DISCUSSION

Allocate costs you the same overhead as running a program except that it does not create a stack or obtain a PCB entry. The advantage to allocate is that you can do the high overhead part of creating a process (searching the SLs and loading the CST and XCST) ONCE and not have to do it again each time you run the program.

Note that allocate is most useful for programs that are run often but not constantly thruout the day. (I.E. if someone else is already running the program then allocate gains you nothing. Likewise it also costs you nothing).

Choose programs to allocate carefully as it will use up table entries, specifically the CST tables. The best choice for allocation are the programs like EDITOR, FCOPY etc. These programs are run often thruout the day but are not likely to be run all day long. User programs for casual inquiry or the like are also good candidates.

ALLOCATION LAB

Do this lab as a class exercise. It involves measuring items that can be affected by other activity on the system so you should try to be as careful as you can to control external events.

1) Determine how you can measure the overhead necessary to run a program. Include CPU time, Elapsed time, and DISC IOs if you can.

2) On a quiet system, measure the overhead to run the BASIC interpreter when it is not ALLOCATED. (You should issue the :DEALLOCATE BASIC.PUB.SYS command to be sure).

3) Repeat the measurements but this time have a second user run BASIC before the measurements start. (Wait until BASIC comes up on the second terminal before starting the test).

4) Repeat the exact same measurements after ALLOCATING the program. :ALLOCATE BASIC.PUB.SYS

RESULTS:

How much does ALLOCATING BASIC save you ?

EXTRA CREDIT LAB

If you have the extra time and inclination you can do the following extra credit work on ALLOCATE:

1) Determine the difference in savings between allocating BASIC.PUB.SYS and allocating FCOPY.PUB.SYS. Why would there be a difference? Can you predict the benefits for allocating COBOLII.PUB.SYS ?

2) How much overhead is there in doing an ACTIVATE vs a CREATE ? (This question will require you to write a program using Process Handling capability). Which part of this process does ALLOCATING help ?, By how much ?

   Hint: Create BASIC.PUB.SYS as a son process using the CREATE intrinsic then use the ACTIVATE intrinsic to start it running. You can call TIMER, PROCTIME, and CLOCK intrinsics in your program to get different types of timings if you desire.

MEMORY MANAGER (MAM)

MAM
The Memory Manager

Q What does a memory manager have to do ?

Q What methods have been used to manage memory ?

Q What method does MPE use ?

Q How does it work and what control do I have over it ?

---

1. keeps things in memory
2. Options

DISCUSSION

MPE divides main memory into two areas. The first, fixed memory, contains only the tables and code that the operating system requires to be memory resident. The remainder of memory, called linked memory, contains all other code and data. User and operating system segments are brought into this area by the memory management routine FETCHSEGMENT as they are required. When there is insufficient main memory to support the workload, the overall performance of the system can be noticeably reduced.

Memory management functions consist of free space allocation, segment replacement and garbage collection.

Free space allocation is required when a segment fetch is to be performed. The free space algorithm selects the smallest hole that is large enough to hold the segment (best fit). A fault can occur on a segment that has been marked to be released from memory but hasn't yet been physically removed. In this case the segment will be RECOVERED.

Segment replacement occurs when free space of adequate size is not available. Segments which have not been accessed recently are marked as OVERLAY CONDIDATEs. When actually removed from memory a segment has been RELEASED. The selection process scans memory on a bank by bank basis in ascending address sequence. The scan is stopped when sufficient space has been found for the segment that caused the fault. Each scan begins where the previous one stopped. When the end of memory is reached during a scan a CLOCK CYCLE is counted.

The variable size allocation policy of MPE tends to produce small, unusable holes scattered throughout memory. Garbage collection attempts to minimize this effect by combining small holes into larger usable holes.

Each time a segment is released an attempt is made to join the new free space with nearby areas that were already free. This is referred to as local garbage collection. Each time the dispatcher fails to find anyone ready to run AND memory is full then garbage collection is begun on a memory wide (global) basis. This activity is stopped as soon as someone needs the CPU.

Local collection occurs on a routine basis while global collection occurs only when the memory manager has recently had problems meeting memory requests.

Requirements of a Memory Manager

* An efficient mechanism and policy for allocation of primary memory among competing processes.

* A mechanism for location and independent referencing of information in primary memory.

  – An extension to that mechanism to allow the concept of VIRTUAL MEMORY.

* An efficient mechanism for file access

PROBLEMS

* Requirements of executing proccesses vary with time and generally are unpredictable.

* A user should be able to program with little or no concern for the size of primary memory.

METHODS OF MEMORY MANAGEMENT

* Primary memory is a limited resource that needs to be efficiently allocated.

* Several mechanisms exist which allow for the reuse of main memory.

* Examples: SWAPPING
  OVERLAYING
  PAGING
  SEGMENTATION

* Many systems utilize memory management algorithms that reflect a combination of the above methods

SWAPPING

* Each process is allocated all of main memory. In order for another process to run the first one had to yield its use of memory. At first this only occurred when the process had terminated. Later, the operating system imposed a QUANTUM of time, after which the process was forced to yield main memory. The process was then SWAPPED OUT to secondary memory (disc) and another process SWAPPED IN.

* ADVANTAGES: Very simple to implement.

* DISADVANTAGES: Transfer time beween primary and secondary memory slowed down overall thruput.

  Swapping increased the use of CPU and DISC which left less for the processes to use.

  Main memory was inefficiently used (Unneeded code and data was present when the entire process was present in memory).

  Only one process is ready to execute in main memory at one time. (No true multitasking could occur).

---

1 Single Process

OVERLAYING

* A process is allocated a main memory area and executes in that area until completion.

* Several modules within a process are permitted to be mapped into the same memory space but only one at a time.
(Assumption: Overlay modules don't need to be in memory at the same time).

* Process modules are dynamically moved in and out of main memory during execution as they are needed. This is usually done under control of the user process itself.

ADVANTAGES: Decreases the amount of main memory required by the programs.

Programs that are larger than main memory can run using an overlaying structure.

DISADVANTAGES: The user has responsibility for managing the overlaying mechanism.

This scheme doesn't allow for the dynamic nature of system workload and can't adjust to it.

PAGING

* Main memory is divided into equal sized blocks.
* The memory needed by a process is divided into equal sized units called PAGES.
* Pages of information are loaded into main memory as they are required.

ADVANTAGES: User transparent

Allows for multiple processes in main memory.

Simple allocation of memory due to fixed length.

No fragmentation of memory space.

DISADVANTAGES: Fragmentation within a program. (One page may not always be a logical program subdivision. This can lead to a phenomenon known as THRASHING where frequent transfers between pages incurs high overhead).

Additional overhead is added in addressing the pages.

Some memory is wasted when the program does not require an integral number of pages. Memory allocation is always in whole blocks.

SEGMENTATION

* An enhanced paging scheme

* Allows the memory requirements of a process to be divided into VARIABLE LENGTH SEGMENTS.

ADVANTAGES: Thrashing is reduced by having memory segmentation match the program's own internal structure. (Less chance to have a loop cause inter-segment transfers).

Facilitates efficient use of main memory (No waste as in a fixed length page scheme) It also allows the memory manager to easily balance a processes use of main memory against its priority and the system load.

DISADVANTAGES: Increased system overhead due to the sophisticated addressing scheme.

DISCUSSION

This is the method of memory management used by MPE. In order to reduce the overhead required to address the segments the HP3000 and MPE have built in hardware and microcode routines. This results in the addressing being as fast as direct addressing but it does add some restrictions as to the size of each segment in a program.

All user programming generates addresses that are relative to the start of a segment (code or data) or are relative to a special purpose register that points within the segment. The HP3000 CPU has a special hardware "preadder" that adjusts references to memory by "preadding" the address to the register contents while the previous machine instruction is being executed. (This is a part of the microprocessor "pipeline" feature on all HP3000 CPUs). As a result, the special addressing required by a segmented memory scheme costs no additional execution time on the HP3000.

WHAT'S IN MEMORY
(From MAM's point of view)

FREE SPACE &
SEGMENTS CODE
DATA
DISC DOMAIN (CDT - Cache Dir Table)

Segments may be in one of three states:

PRESENT: The segment is in main memory and ready for use.

ABSENT: The segment is not in main memory (It is on disc instead)

OVERLAY CANDIDATE: The segment is in main memory but has been selected to be overlayed. It has been copied to disc (if necessary) but can be RECOVERED and used in main memory if needed.

In looking for room in main memory to put a new segment, overlay candidates are considered as FREE SPACE.

MEMORY REGIONS

In order to more efficiently handle the task of memory management, memory is grouped into REGIONS. A REGION is a collection of contiguous segments in memory that share a common condition.

CONDITIONS:

AVAILABLE; This region of memory can be used to allocate new segments whenever needed. This area consists of FREE SPACE and OVERLAY CANDIDATE segments.

RESERVED; An available region that has been selected for use. Not all overlay candidates have been completly written to disc yet and/or the new segment hasn't been fully read in from disc yet.

ASSIGNED; The region contains only segments marked as PRESENT (ready for immediate use).

Generally. Region containing 1 segment except for free space

ATTRIBUTES OF THE MEMORY MANAGER

* Integral part of the scheduler.

* It considers system requirements for memory as a whole, not just for one process.

* It opens up space in neighboring areas of memory whenever possible to maximize the size of the largest available region.

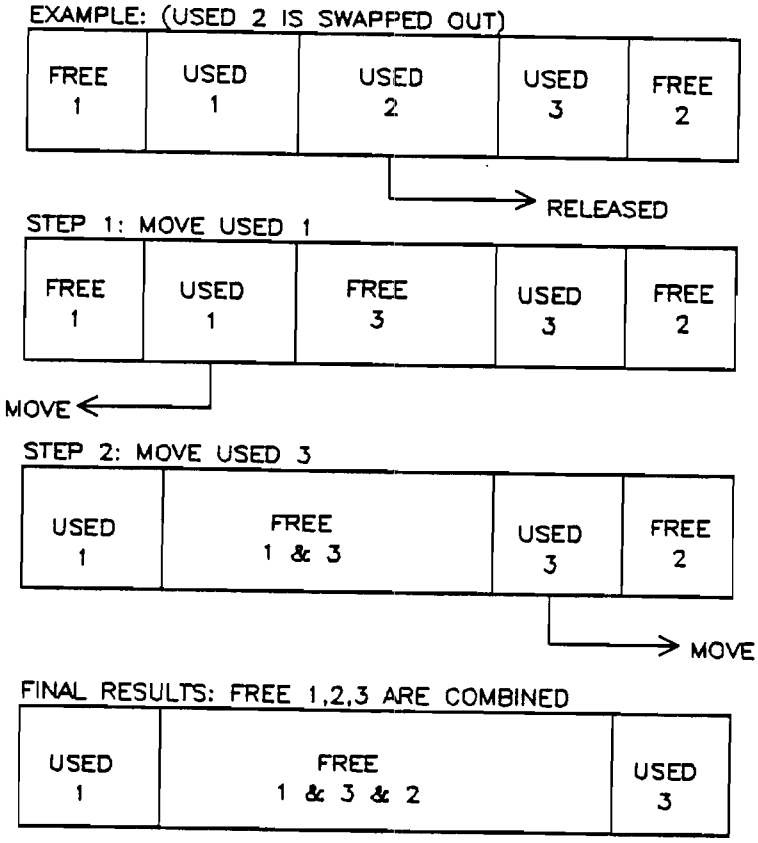
* It can process more than one request for memory at a time, in parallel operations.

* It does anticipatory writes of overlay candidates as low priority background to minimize overhead

* It now uses a "BEST FIT" algorithm rather than a "FIRST FIT" technique as this has been found to be better with larger memories.

* Garbage collection is done locally whenever a new available area is created and globally whenever the CPU would have nothing else to do and memory is scarce.

GARBAGE COLLECTION



WHEN IS GARBAGE COLLECTION DONE ?

1) LOCAL garbage collection is done whenever a segment is marked to be swapped out. (Made an overlay candidate).

2) GLOBAL garbage collection is done whenever the scheduler would PAUSE (no process can use the CPU) and "MEMORY PRESSURE" exists. (There is a general shortage of memory).

GLOBAL garbage collection attempts to increase the size of the largest free area first then the next largest etc. In this case more than one segment can be moved to make room.

Global garbage collection goes on until either no more benefit can be obtained from it or until a process needs the CPU. (Global garbage collection can be interrupted at any time).

WHEN DOES MEMORY MANAGEMENT OCCUR ?

1) At initial process launch and termination.

2) Subsequent process launches if needed segments are absent. (Based on the MINIMUM LOCALITY).

3. Whenever an ABSENT segment is referenced.
   PCAL and EXIT
   STACK/DATA SEGMENT move

4) Data segment expansion or contraction
   NO ONE KNOWS HOW MANY times

5) FREEZE and LOCK requests

DISCUSSION

1) The first time a program is given the CPU none of its segments will be in memory. MAM is invoked to make this processes MINIMUM LOCALITY present in memory. At process termination the segments for this process must be destroyed and their memory returned to free space. NOTE: this activity can cause memory management activity even when plenty of free space is available. Beware of calling a system memory bound just because of high MAM activity. If programs are being run and terminated rapidly then MAM can get real busy even if plenty of memory is available.

2) MINIMUM LOCALITY is the smallest collection of segments which must be in main memory in order for a process to execute and consists of:

   The CURRENT code segment
   The DATA STACK
   and possibly ONE EXTRA DATA SEGMENT (if the DB register is pointing to it).

3) The memory manager does not have to constantly check whether or not it should swap something in (like it does on fixed page systems). The only time MAM may be needed is when the user program addresses OUTSIDE the current segment. For code segments this is when a subroutine is called (PCAL) or returned from (EXIT). For data segements this is one of only a few machine instructions that transfer data between extra data segments (MFDS, MTDS etc).

   This means that most of the HP3000 machine instructions don't have to worry about whether the data they need is in memory or not. It ALWAYS WILL BE (since a process won't get the CPU unless its minimum locality is in memory and most instructions can't address outside of the minimum locality). This reduces the time spent in each of these instructions significantly.

4) In order to expand a data segment it must be swapped out to disc (virtual memory area) where enough space has been set aside to expand it to its maximum size. It is then expanded on disc and swapped back in to main memory. This process can be time consuming so it should be done thoughtfully i.e., overuse of the ZSIZE intrinsic can cause performance problems.

5) Freeze and locks are discussed more fully later.

HOW DOES MAM DO ITS JOB ?

MAM gets involved for "SPECIAL REQUESTS" (Expand, freeze etc.) and to make a segment present in memory.
The current algorithm for memory allocation is as follows:

1) Is the segment a recoverable overlay candidate ? If so then mark it "PRESENT".

"RECOVERED OVERLAY CANDIDATE"

2) Check the system global area for the size of the largest available region. Will this segment fit into this region ? If so then "FOUND FREE SPACE"

2A) Locate the size of the smallest available region that is big enough to hold this segment using the AVAVAILABLE REGION BIT MAP. ("BEST FIT ALGORTHM").

2B) Using the AVAILABLE REGION LIST, locate the first region of this size.

2C) Mark the region as "RESERVED".

2D) Return the unused portion of the region to the AVAILABLE LIST.

2E) Make sure all IOs from this region are completed. (Raise backgroung IO priority to HIGH priority if necessary).

2F) Read the desired segment into this region.

2G) Update the tables and mark this region as "ASSIGNED". Mark the segment as "PRESENT".

PRG MPS IE

---

2 Types of Special Requests - Spec Request TABLE
Expand Segments
Freeze Memory.

1. Ignore "Assigned"
2.

3) If no available region is big enough to hold this segment then try to make larger available regions.

3A) Scan thru memory, region by region.

3B) When an assigned region is found,

If this region has not been referenced lately, then MAKE SEGMENTS INTO OVERLAY CANDIDATES

If enough room now, then go to step 2C.

If not then do LOCAL GARBAGE COLLECTION in this region.

If enough room now then go to step 2C.

3C) Go on to the next region in memory. If we have gone all the way thru memory then GIVE UP

For now, try this segment later.

MEMORY PRESSURE

WHAT IS IT ?

Memory Pressure is an additional way to prevent the memory manager from THRASHING. (Spending too much time and resources swapping things in and out of memory).

WHAT DOES IT DO ?

It changes the rules that the memory manager uses to satisfy requests for room in memory. In essense it changes the memory manager so that it will not try so hard to bring new things into memory.

In this way segments that are in memory are not thrown out before they have a chance to get some work done.

HOW IS IT SET ?

The total CPU time used while MAM cycles through all banks of memory to create overlay candidates is compared to the "MINIMUM CLOCK CYCLE" value. If memory is cycled through in less than the minimum clock cycle interval then MEMORY PRESSURE is set ON. The default minimum clock cycle is one second.

NOTE: The ONLY time GLOBAL GARBAGE COLLECTION is done is when MEMORY PRESSURE is SET ON.

FROZEN SEGMENTS

FREEZING a segment in memory insures that it can not be swapped out or moved from its location in memory. Requests to freeze a segment will not move the segments before freezing them.

Freezing is generally used by the IO system because a segment can't be allowed to move while IO is in progress to or from it. (The IO hardware only uses absolute memory addresses).

Too much freezing can fracture main memory because the segments will be frozen wherever they happen to reside at that moment.

Normal users can not freeze their own segments. This function can only be requested by the operating system (or priv mode users).

LOCKED SEGMENTS

Locking a segment into memory is like freezing in that such a segment can not be "swapped out". Unlike freezing, locking allows a segment to be moved as long as it remains in memory.

Locking is usually used for subsystem interrupt handlers that can not stand the length of time a swap takes. (Notably the communications system locks its code into memory).

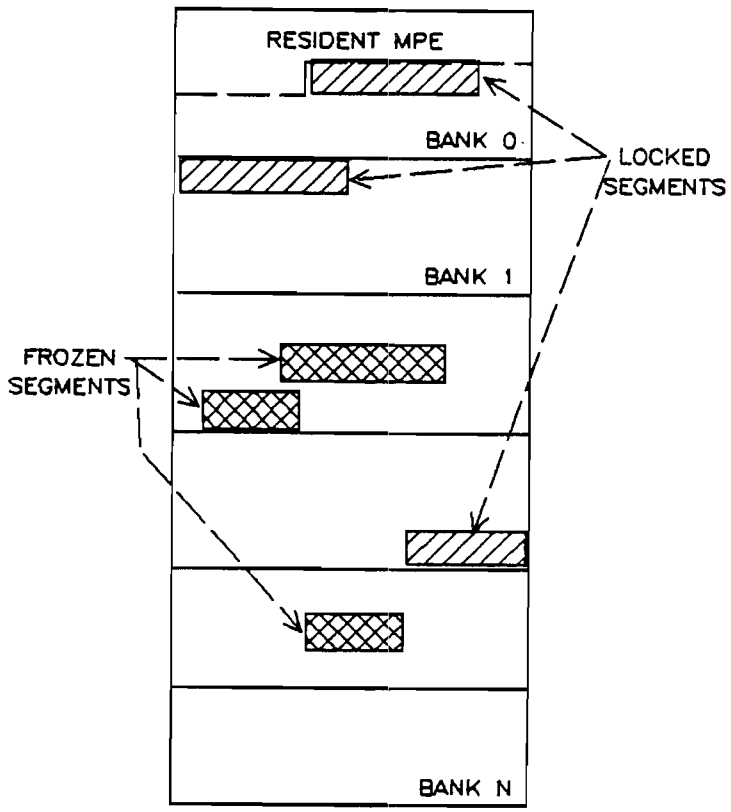
? Why not make these segments a part of memory resident MPE ?

An attempt will always be made to move a segment to a bank boundary before it is LOCKED. This is to reduce the chances of fragmenting main memory.

LOCKING involves high overhead because an assigned region or a frozen sgment might exist at the bank boundaries. Substantial reorganization of main memory may be required to move the segment.

LOCKING, like FREEZING, can only be requested by the operating system or priv mode user programs.

LOCKED & FROZEN SEGMENTS



MEMORY MANAGER LAB

RUN OPT.PUB.SYS

1) Using the GLOBALS display; How can you tell if memory pressure exists ?

2) Go to the Memory Contents Display **(#MM)**. Can you find any FROZEN or LOCKED segments? How can you tell if a segment is FROZEN vs LOCKED ?

3) Go to the CPU-Memory Management Report **(#CR)**.

4) What is the difference between process launches and swap-ins ? Which one is bigger ? Will this always be the case ?

5) How often is MAM running ?

6) Does this system have enough memory on it ? How can you tell ?

7) Go to the Memory Management Activity display. **(M)**.

8) Can you tell how many times MAM marked a segment as an overlay candidate then had to change its mind ?

9) How many special requests are there ? What could they be for ?

10) Can you tell how close this system is to Memory Pressure ? How ?

11) Which is greater, MM Reads or MM Writes ? What can you tell from this ?

SCHEDULER AND DISPATCHER

A HISTORY OF SCHEDULERS

* The first scheduler of note was a card reader. It forced a simple but effective FIRST–IN FIRST–OUT scheduling of programs. Each program had 100% of the system resources while it ran. Anything it didn't use was wasted.

* Later main memory was PARTITIONED so that more than one program could reside there at the same time. Now when one program no longer needed the CPU (like when it was waiting for IO to complete) the CPU could service another program in memory. One problem: If a program went "CPU BOUND" then no CPU time was available for the others. (Enter the first system HOG).

* Enter the new and improved TIME SLICE SCHEDULER. This used memory partitions as above but with a twist. Now as each program got the CPU a timer was started. If the program had not yielded the CPU before the timer went off then the CPU was forcably removed and given to another program. This is the first use of a TIME QUANTUM for scheduling.

Everything was humming along real good now. All programs got a fair shake at getting the CPU. There was little wasting of resources and life was good in the DP world.

BUT, (you knew there would be a but, didn't you?), now a new requirement was added. The boss found out that payroll was always late and wanted to know why. What he found was that other programs were slowing it down. So much for equal access to the CPU. What was needed now was a way to PRIORITIZE the programs so the important ones got special treatment.

STATIC PRIORITIZATION was first developed. This assigned a priority to each program on the system. Any program with law priority had to wait until no higher priority programs needed the CPU before it got it.

DYNAMIC PRIORITIZATION came next. This is where the priority of a program was set to a RANGE. As the program ran, its actual priority was adjusted within that range based on its use of the system resources. This scheme proved better at balancing the system's allocation of resources to the dynamic load imposed upon it.

POLICIES OF THE MPE SCHEDULER

* All decisions about CPU scheduling for a READY process are made by the scheduler.

* The scheduler will allocate the CPU to the highest priority process that is READY to run.

* The scheduler will maintain a DISPATCH QUEUE of all READY processes in order of priority.

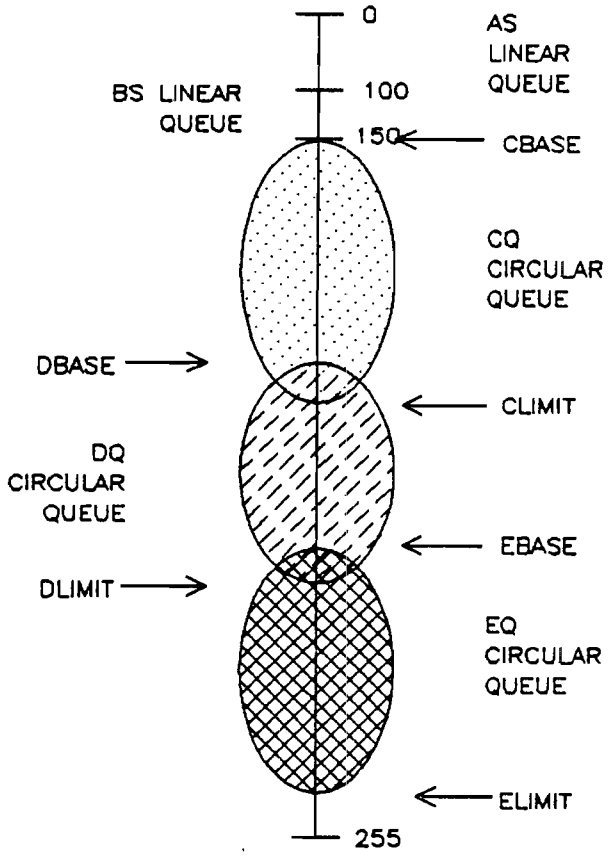
* a process becoming READY will be placed into the DISPATCH QUEUE in order of priority.

* A higher priority process coming READY must be able to preempt a lower priority process in execution.

* CPU BOUND processes, (Those that block infrequently), must not be able to monopolize the CPU, regardless of priority.

The Dispatcher decides what the CPU will do next by inspecting the priority ordered queue of processes requiring CPU service. When a process is encountered which is ready to run, it is launched. If the process requires some memory scheduling, the procedure SWAPIN is called directly by the dispatcher. If there's nothing better to do and free memory space is short, main memory garbage collection takes place.

MPE SCHEDULING QUEUE



QUEUE CHARACTERISTICS

AS LINEAR QUEUE; PRIORITY RANGE 0–100
- –Highest available queue in system
- –No priority adjustments
- –Reserved for MPE

BS LINEAR QUEUE; PRIORITY RANGE 101–149
- –No priority adjustments
- –Some MPE processes
- –Reserved for high priority user processes

CS CIRCULAR QUEUE; PRIORITY RANGE ADJUSTABLE
- –Dispatcher controls priority
- –Transaction oriented processes
- –Considered "Time Share" queue

DS CIRCULAR QUEUE; PRIORITY RANGE ADJUSTABLE
- –Dispatcher controls priority
- –Considered "Batch" queue

ES CIRCULAR QUEUE; PRIORITY RANGE ADJUSTABLE
- –Dispatcher controls priority
- –Considered "Background" queue

CS QUEUE PRIORITY ADJUSTMENTS

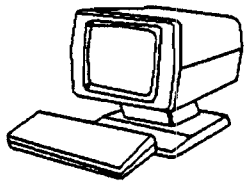
* CS queue priority adjustments are primarily based on an "AVERAGE SHORT TRANSACTION" time and memory faults.

The AVERAGE SHORT TRANSACTION time is calculated by averaging the last 100 "short" (less than 32 second) terminal transactions. The transaction is measured as CPU time between terminal reads.

Any CS priority process that uses more CPU time than the AVERAGE SHORT TRANSACTION will have its priority lowered. This adjustment is made each time the process expends one time quantum.

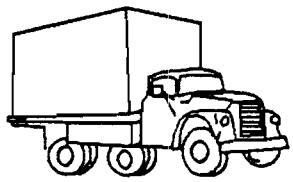
Any CS priority process that suspends due to needing a segment that is not in memory will also have its priority lowered.

When a CS priority process does a READ on a terminal device then its priority will be raised to CBASE.



DS and ES PRIORITY ADJUSTMENTS

When a DS or ES priority process uses one quantum of CPU time its priority is reduced.



TEMPORARY PRIORITY ADJUSTMENTS

Whenever a process holds a resource that is impeding a higher priority process then it will temporarily have its priority raised to that of the higher one until it can release the resource.

TUNE COMMAND

```
                                CQ
:TUNE minclockcycle ;DQ = base , limit, min, max
                                EQ
```

minclockcycle	Minimum value (in milliseconds) for replacement algorithm cycle through memory. If MAM cycles through memory in less than time value, MEMORY PRESSURE exists.
base	Priority at which C, D, or E processes will begin in queue. Also the highest priority they can normally obtain.
limit	Worst (highest number) priority a C, D, or E priority process can normally attain.
min/max	For C processes, this is the range than the AVERAGE SHORT TRANSACTION value can attain. For D, and E processes the "min" is the time quantum that must be used before priority is reduced. "max" is not used.

DISCUSSION

The **TUNE** command changes the filter or priority limits of a circular subqueue, and is used primarily in on-line tuning of the system to best accomodate the current load.

CAUTION! MISUSE OF THIS COMMAND CAN SIGNIFICANTLY DEGRADE SYSTEM OPERATING EFFICIENCY!

A CS process is given a priority of CBASE when it begins. When a process stops (for disc I/O, terminal I/O, preemption, etc.), a new priority is determined so that it may be re-queued for the cpu. If the process has completed a transaction (a transaction is defined as the time between terminal reads), the priority become CBASE. The value of an "average short transaction" is then recalculated. If the CS process has not completed a transaction, and if the process has exceeded the average short transaction filter value since its priority was last reduced, the priority is decreased by 2. Also, if the process stopped because it needed memory resources, its priority is decreased by 1.

DS and ES processes begin at DBASE and EBASE respectively, and are re- scheduled according to the same criteria as used for CS processes, with the exception that a fixed value (the value of max which has been specified for the subqueue) is used in place of the average short transaction value, which is used for CS processes only.

If the values specified for max in the CS, DS, and ES queues are too large, system response may become erratic. If they are too small, excessive swapping may result because process-swapping occurs too frequently. In either case, system performance is degraded. Recommended settings for min and max for the CS subqueue are 0 and 300, as it is desirable to favor short transactions in most cases. A value for min and max of 10,000 usually produces efficient system operation for the DS and ES subqueues.

The minclockcycle parameter is used by the memory manager to determine when thrashing (excessive memory management activity) is occuring. Making this value smaller increases the probability of thrasing. Recommended value for this parameter is 1000.

PROCESS QUIESCE

When does a process give up the CPU?

- When it blocks naturally (for IO etc).
- When it is preempted by a higher priority process.
- Whenever its time quantum expires.

How does the dispatcher know a quantum has expired ?

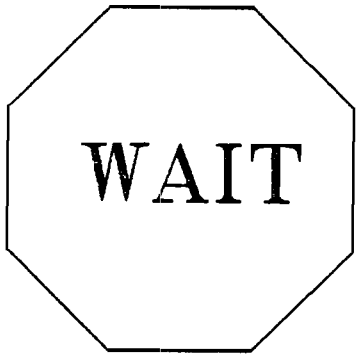
- The system clock is set to interrupt the CPU.
- the procedure TICK notifies dispatcher.

How does TICK really work ?

- the system clock interrupts every 100 ms
- Each time it does TICK is called.
- When TICK has run three times it does a DISP instruction which invokes DISPATCHER.

How much time does a process really get ?

Since the process may start at any time with respect to the system clock, it can get from 201 to 299 ms (three clock ticks) before the DISPATCHER gets control again. The DISPATCHER then decides whether to continue with the current process, adjust its priority, or to schedule a higher priority process.



– WAIT is called whenever a process must stop running until something else occurs.

– A parameter is passed to WAIT indicating what event the process is waiting for. This parameter is stored in the PCB. Special care is taken so that if the event has already occured then the process is not actually suspended at all but rather goes right on processing.

WAIT CONDITIONS

MOURNING: A parent process is waiting for
a child process to die.

GLOBAL RIN: Waiting for a Global RIN to be locked

LOCAL RIN: Waiting for a Local RIN to be locked.

MAIL: Waiting to receive mail from a relative.

BIO: Waiting for BLOCKED IO to complete
This is normal IO.

IO: Waiting for NO WAIT IO to complete

UCOP: Waiting for the User COntroller Process
(The parent process of all Command Interpreters)

JUNK: None of the above or below. Used by
System Processes.

MSG: File system IPC message file wait.

IMP: Process is IMPEDED (usually for another
process to release something like
a Data Base Control Block).

SON: Waiting on activation from child process

FA: Waiting on activation from parent process.

TIM: Waiting on the system clock (timer)

MEM: Waiting for memory

CPU STATES

BUSY: Running a process

PAUSED FOR DISC: The CPU is PAUSED and at least one USER DISC IO is going on.

PAUSED FOR SWAP: The CPU is PAUSED and at least one MAM DISC IO is going on.

PAUSED FOR DISC & SWAP: The CPU is PAUSED and at least one USER and one MAM DISC IO is going on.

IDLE: The CPU is PAUSED and no DISC IO is going on.

BACKGROUND GARBAGE COLLECTION: The CPU would be paused but due to MEMORY PRESSURE it is doing global garbage collection.

MEMORY ALLOCATION: MAM is working on satisfying a request for memory.

ICS OVERHEAD: The CPU is busy handling interrupts (It is using the Interrupt Control Stack).

Under what circumstances is the CPU not executing code? (i.e. is PAUSED)

When is the CPU marked as IDLE ?

What states can the CPU be in if a batch job is running ?
(HINT: Input is from a disc spool file, output is to a disc spool file)

DISCUSSION

The CPU is actually executing the PAUSE instruction whenever the CPU state is:
PAUSED FOR DISC
PAUSED FOR SWAP
PAUSED FOR DISC & SWAP
IDLE

It is marked as IDLE only when no CPU or DISC IO activity is going on.

If a batch job is running then normally the CPU will never be in the IDLE state since the normal reasons a batch job will suspend are to get a new command from $stdin (a DISC IO) or to write data to $STDLIST (another DISC IO). For terminal accesses where $stdin and $stdlist are not a DISC then the CPU will show IDLE time.

Exceptions to this rule are whenever the batch job does NON DISC IO (such as mag tape) or suspends for another reason (like waiting on a REPLY from the console).

If a batch job is running it is normal to see no IDLE CPU time. This doesn't mean that the system is CPU BOUND. (Don't forget that the CPU is actually not being used when it shows up as PAUSED FOR DISC, SWAP or DISC & SWAP).

DISPATCHER LAB

RUN OPT.PUB.SYS

1) Go to the CPU Usage Display **(#CC)**. Explain the current CPU state.

2) What causes a process launch ?

3) Why are process preempts less than launches ? Will this always be true ?

4) Look at the CPU state information.

   What would you expect to see if this system was bottlenecked

   on CPU ?

   on MEMORY ?

   on DISC IO ?

5) What does the "Average CPU Time per Transaction of Interactive Processes" relate to ?

DISCUSSION

A process launch occurs when the dispatcher gives the CPU to a new process. Any time a process has been suspended, for any reason, and is then given the CPU again, that is a process launch. There is also an INITIAL PROCESS LAUNCH that occurs when the process is first started.

Each time a process is preempted (suspended so a higher priority process can run) there is a corresponding process launch (of the higher priority process). In addition, processes suspended for other reasons (IO, etc) will also be launched whenever their reason for suspending has been satisfied. Thus the total number of launches should always be greater than or equal to the number of preempts.

If a system is bottlenecked on CPU then the amount of time the CPU was paused should be small. (This includes IDLE, PAUSED FOR DISC, PAUSED FOR SWAP, and PAUSED FOR DISC & SWAP). The CPU BUSY and ICS OVERHEAD should be most of the CPU TIME.

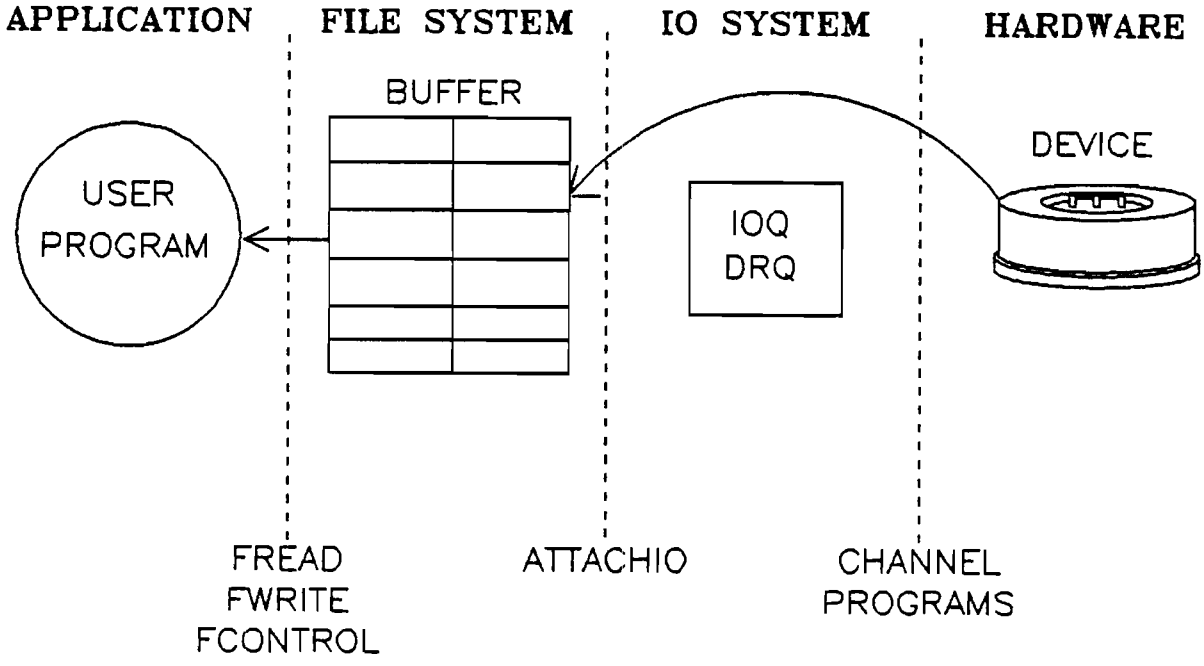
If the system is MEMORY bound the PAUSED FOR SWAP and MAM will be consuming a large portion of the CPU's time. BUSY time should be small as compared to MAM and PAUSED FOR SWAP. Note that the only time the CPU is doing useful work on the user's processes is when it is in the BUSY state.

A DISC bound system will show a large percentage of time in the PAUSED FOR DISC, PAUSED FOR SWAP, PAUSED FOR DISC & SWAP and ICS OVERHEAD categories.

Notice: the ICS OVERHEAD value will normally increase as DISC IO increases (up to about 20% of the CPU). This is because the CPU must handle the high rate of interrupts generated by the disc drives under heavy load. If the ICS OVERHEAD is high and the disc IO rate is low then something else is interrupting the CPU. (Notably this could be a runaway IO device such as a streaming modem or hung Cartridge Tape).

The AVERAGE CPU TIME PER TRANSACTION value is used to calculate the average short transactiom time which controlls the CS QUEUE priority adjustments. It is the amount of CPU time used by interactive processes between reads to the terminal. (Not counting any transaction over about 32 seconds long, like compiles).

IO SYSTEM



INTERFACES

DISCUSSION

The previous page shows the elements involved in doing a user IO. Specifically it shows a BUFFERED read from disc.

The interfaces are ridgidly defined so that each layer of MPE can be maintained by a different group of people. It's too big for any one person to handle completely.

1) The user application issues a request to get a record from the device. This can be thru a COBOL ACCEPT, or standard input routine, FORTRAN READ statement or any such method. The various languages will all eventually end up calling a standard file system intrinsic such as FREAD. The user application can also call the file system intrinsics in some cases. This will eliminate the language overhead. The FORTRAN formatter for example has considerable overhead in it.

2) The file system then handles such things as SECURITY, FILE ACCESS MODES, and BUFFERING. In this case (a FREAD) the file system will check to see if the record requested is present in the file system buffer (an XDS obtained by the file system on the user's behalf). If so then the file system simply copies the record to the user's stack and the request is complete.

   If the record is not in the buffer then the file system calculates which BLOCK from the device contains it and then issues a call to ATTACHIO to read that block into the buffer. (translation from file name to logical device to physical device is done at this time).

3) Now the IO system gets the request. All ATTACHIO does at this time is to add an entry into the IOQ table (or DRQ for discs). This entry contains all the information necessary to perform the actual physical data transfer.

   Once the entry is added to the IOQ a quick check is made to see if the IO PROCESS MONITOR servicing this device is currently doing another IO for this device. If so then the new request is QUEUED behind the others so the monitor will start it once the others are finished.

   NOTE: As of MPE IV the DISC IOQ entries are chained together based on the CPU PRIORITY of the requesing process. This is to prevent low priority batch jobs from monopolizing the IO capacity for the disc drives. All non-disc devices are queued on a first come first served basis.

   If the device being referenced is NOT busy doing an IO then the process monitor is "POKED" so that it will wake up and start servicing this request immediatly.

   At this point ATTACHIO will decide whether to allow the user process to continue processing or not. If the request is a normal one then the user process is BLOCKED FOR IO. If the user is using NO WAIT IO then it is allowed to continue processing while the IO is accomplished.

4) Once the IO PROCESS MONITOR is servicing a request from the IOQ (or DRQ) it decodes the logical device information all the way down to the DRT and UNIT for the device. It then calls the proper IO DRIVER (from the IO configuration) and notifies it that it has an IO request to handle. This is all done thru several IO tables such as the Device Information Table (DIT).

5) The IO DRIVER looks at the request then builds a CHANNEL PROGRAM in a special area of the DIT. These channel programs are not HP3000 machine code but rather a special code used by the IO HARDWARE. The instructions can be from 32 to 128 bits long.

6) Now the driver issued a special machine instruction to transfer control from the CPU to the IO hardware. On a series I,II,III this is the Start IO (SIO) instruction. On all HPIB systems it is a Start IO Program (SIOP) instruction. As a part of this instruction the controller address (DRT) and the location of the channel program is passed to the IO hardware. On a series I,II,III this is the IO Processor. On the HPIB systems, this is the General IO Channel (GIC).

   From this point on the CPU is freed to do other things while the IO hardware processes the channel program.

7) Under control of the channel program the IO device does its thing. (SEEK, LOCATE DATA, READ DATA INTO FILE BUFFER, RETRY the transfer if trouble, REPORT COMPLETION STATUS etc.). Once the device has done all it can do it then asks the GIC or IOP to INTERRUPT THE CPU.

8) The CPU sees the IO interupt and "warps out" of whatever it is doing. A special bit of code called an INTERRUPT HANDLER now takes over. It uses the Interrupt Control Stack (remember the previous section that showed when the CPU was using this stack?) The function of an interrupt handler is necessarily short & sweet. All it has to do is figure out why the interrupt has occurred then schedule the proper IO MONITOR to take over from here. This is so the CPU can quickly get back to what it was doing before it was interrupted (including running another driver or IO process monitor).

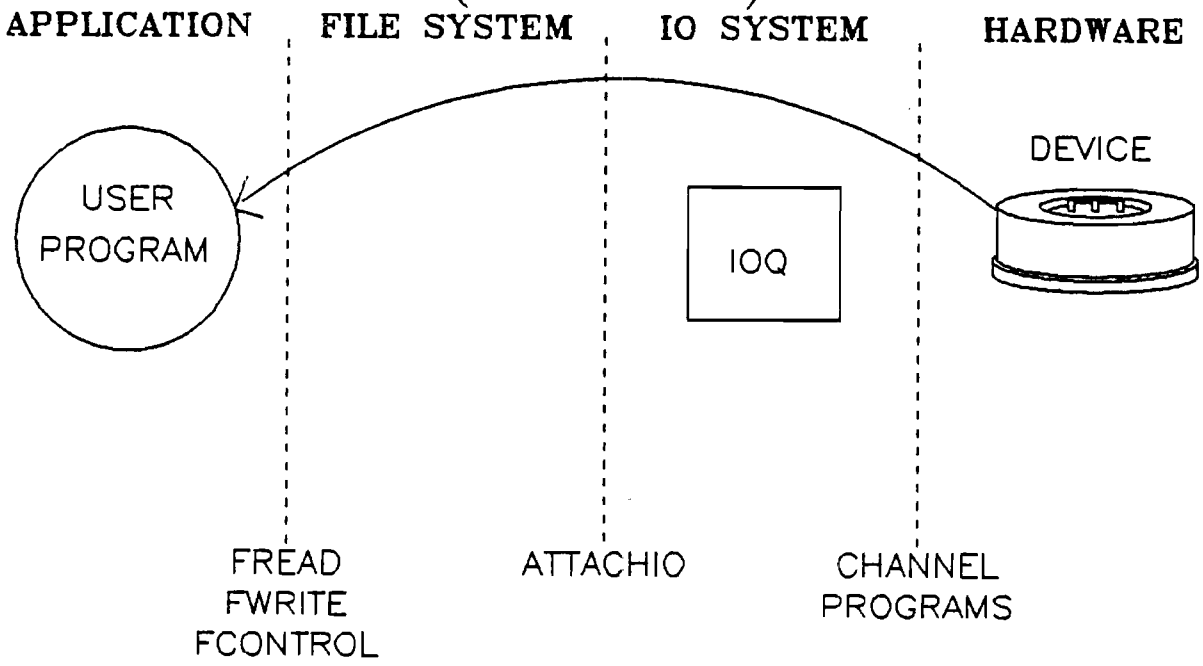
9) The IO process monitor is eventually scheduled (rather quickly actually since they run at high priority) and does the "CLEAN UP" work on the IO transfer. It moves the completion status from the DIT into the IOQ so the user process can get at it then looks to see if another IO request is in the queue for this device. If so it starts the next request. If not it suspends. Before it goes away it will reactivate the user process if it was BLOCKED ON IO or IO waited.

10) The user process now picks up where it left off (Usually right in the middle of ATTACHIO). ATTACHIO returns the completion code then exits back to the file system.

11) The file system then checks to see that the transfer was completed sucessfully. If so it knows the desired block is now in its buffer so it copies the record to the user's stack like usual and allows the user process to continue. If the transfer had an error the file system returns this error condition to the user's process for it to handle.

12) Finally, the user process has the record it desired (or a status saying it couldn't get it) and continues merrily on it's way. This whole process is

IO SYSTEM (NOBUF)



INTERFACES

DISCUSSION

This diagram is very similar to the last one except that NOBUF IO is being used. The major difference is that the file system does not use an extra data segment to hold the block from the file before transfering it to the user's stack.

The physical IO is done from the device directly into the user's data stack by the channel program. Note that this means the user's stack had to be FROZEN in memory while the physical IO was being done. Previously the user's stack could be swapped out while the IO was being done to the XDS. Since a file system buffer is ususlly much smaller than a user stack then the amount of memory fragmentation is less when buffered IO is being done.

Another "feature" of NOBUF IO can be illustrated here. Since the IO device can only transfer data starting on a BLOCK boundary, not a RECORD boundary, all NOBUF IOs will start on block boundaries. If the file is blocked at one record per block then this is no big deal. If, on the other hand, the file has more than one record per block, the user program must be prepared to handle multiple records at a time and do its own deblocking.

If the file is BUFFERED then the file system can DEBLOCK it in the buffer area so the user only has to handle one RECORD at a time, regardless of the blocking factor.

Note the tradeoffs between using BUF and NOBUF IO.
NOBUF freezes the entire data stack in memory whereas BUF only freezes an XDS.

BUFFERED adds additional time while the file system copies the records from the buffer into the user's stack.

NOBUF requires the user program to do their own deblocking.

One last feature of BUFFERED IO is ANTICIPATORY IOs. the file system is smart enough to know when a user, doing serial IO has accessed the last record in a block. When this occurs the file system issues an ANTICIPATORY IO to flush this buffer (if doing writes) or fill the next block (if doing reads). This IO is allowed to go on IN PARALLEL with the user process.

If the file system buffer can hold at least two blocks of data (the default) then this gives the IO system one whole blocks worth of time to get the next block transferred before the user needs it. In many cases this is enough time so the user program will never have to wait for the physical IOs to complete. This is like doing NO WAIT IO without all the hassle.

IMPORTANT NOTE: ANTICIPATORY IOs can ONLY be done when the file if BUFFERED. NOBUF IO does NO ANTICIPATORY IOs. This fact can offset the gains of NOBUF IO in many cases. In short, the file system defaults are most often the best you can do for performance. Special cases may benefit by using NOBUF IO, but the user must understand when to use it and when not to. The best case to use NOBUF IO is when the user program also does MULTI-RECORD transfers and reads many blocks into the stack with a single IO. The user program must then deblock the data locally.

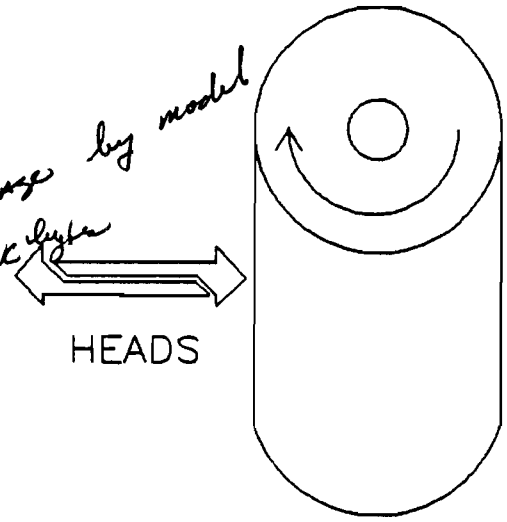
An example: By converting the MPE LOG FILE UTILITY programs from using default buffered IO on the MPE LOG FILES (short variable length records), gains of 10-20 times the performance were obtained. This did involve substantial recoding so that these programs could handle the variable length record formats but the performance was worth the effort.

DOING ONE DISC IO

SOFTWARE SETUP	0–6ms
+ SEEK	0ms
+ WAIT	25ms
+ ROTATIONAL LATENCY	8–11ms
+ XFER DATA	4ms
+ SOFTWARE CLEANUP	0ms

average by model

to give 4K bytes



---

wait : allowing for mult disc drives on some GTC to transfer data

DISC CHARACTERISTICS

DISC TYPE	---SEEK TIME--- MIN	AVE	MAX	ROTATIONAL LATENCY	TRANSFER RATE PER SEC/DATA
7906,7920	5ms	25ms	45ms	8.3ms	737K
7925	5ms	25ms	45ms	11.1ms	737K
7911,7912	5ms	27ms	50ms	8.3ms*	983K
7914	6ms	28ms	52ms	8.3ms*	983K
7933,7935	5ms	21ms	39ms	11.1ms*	1060K

* The PEP firmware reduces the average rotational latency for CS'80 discs if the transfer is for 4KBytes or less.

The typical data sheet includes three very important performance factors: average random SEEK time, average random rotation LATENCY and TRANSFER rate. The table above lists the specifications for the HP disc products commonly found on the 3000 family.

DISCUSSION

Average random LATENCY is fairly straight forward. The time listed is equal to one-half the precisely controlled rotational period. When the disc head arrives over the desired track it can be anywhere from 0 sectors to a full rotation away from the starting sector of target block of data. The random average is one-half of a full rotation. The assumption of random arrival turns out to be very close to what happens in real life.

Average random SEEK time is defined to be the time it would take to do all possible seeks, divided by the total number of seeks. Seek times typically do not include the verify operation. When the head arrives on track, the controller reads at least one sector header to determine that it is on the correct track.

Seek times listed on data sheets are usually higher than what is found on typical systems. In most systems LOCALITY reduces seek times below those shown on the data sheet. Locality means that the head doesn't move as far as the random model would predict. As a rule of thumb actual seek times are about 60% of the seek time on the data sheet.

While TRANSFER rate seems straightforward there are some gotchas. The rate cited might be one of three possible. The first and highest is the bit transfer rate. This is also called the burst transfer rate. The second is the rate that sector of DATA comes off the disc. This rate is lower because not all bits on the disc is user data. Some area is reserved for other functions such as the address header error correction code, etc. HP data sheets show the rate for transferring one sector of data and overhead.

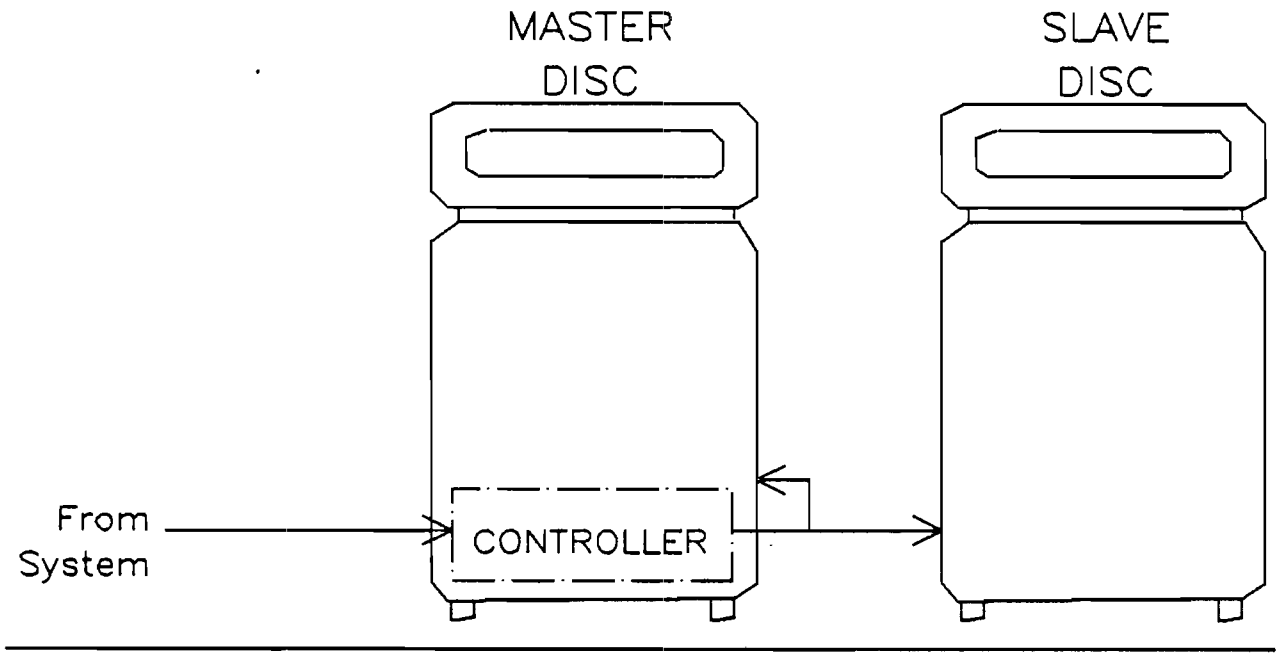
The third rate, and the most important, is the average transfer rate for an entire volume. This rate takes into account the time for doing head and cylinder switches. This is a more realistic transfer rate because files are placed across these boundaries and the actual rate in a user system will statistically tend toward the full volume rate.

From this discussion we can develop a modified "data sheet" whichs account for LOCALITY, the VERIFY time associated with the SEEK, typical SEEK times and full volume transfer rates.

MODIFIED DISC "SPECIFICATIONS"

HP Product	Typical SEEK + VERIFY with 50% locality	average latency	Transfer rate (full volume)
7911	13.5 ms	8.3 ms	770 Kb/sec
7912	13.5 ms	8.3 ms	815 Kb/sec
7914	14.0 ms	8.3 ms	815 Kb/sec
7920	12.5 ms	8.3 ms	615 Kb/sec
7925	13.0 ms	11.1 ms	665 Kb/sec
7933/35	11.0 ms	11.1 ms	892 Kb/sec

MASTER & SLAVE DISCS



DISCUSSION

This slide indicates how slave discs are used. A single disc controller can only TRANSFER data to one disc drive at a time. Unless some additional tricks are used then adding a SLAVE disc to a system will not increase disc performance substantially. (Neglecting the effect of minimizing seek times on single disc systems).

When figuring the maximum number of IOs per second you only count MASTER disc drives in most cases. Exceptions are the series II, III systems which have instituted LOOK AHEAD SEEKS. (a controller can issue a SEEK command to one disc then access another one while the first one is moving its heads but it still can't TRANSFER to more than one disc at a time).

How can you tell how many master discs are on a system without tearing the hardware apart ?

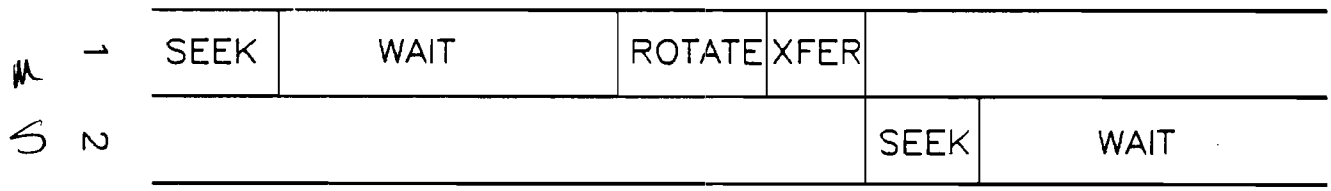
Examine the IO configuration. Each master disc (controller) has a unique DRT number so simply count DRTs on the disc drives.

Only the MAC family of discs (7905, 7906, 7920, 7925) allow the use of SLAVE discs. The main reason for doing this was the fact that the series I,II,III systems could only support ONE MASTER DISC properly and the fact that slave discs cost less.

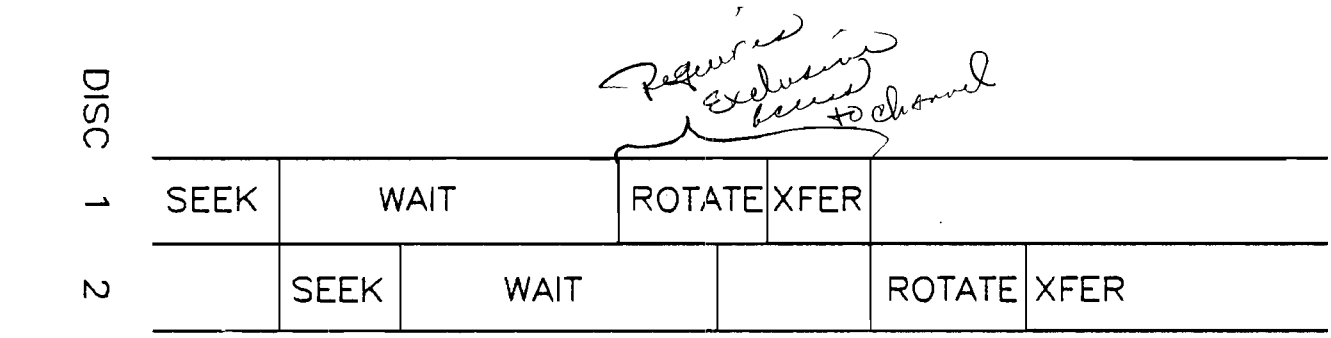
The Command Set '80 (CS'80) disc drives (7911, 7912, 7914, 7933, 7935) are all master discs.

SIII IMPact C.00.08

LOOK AHEAD SEEKS

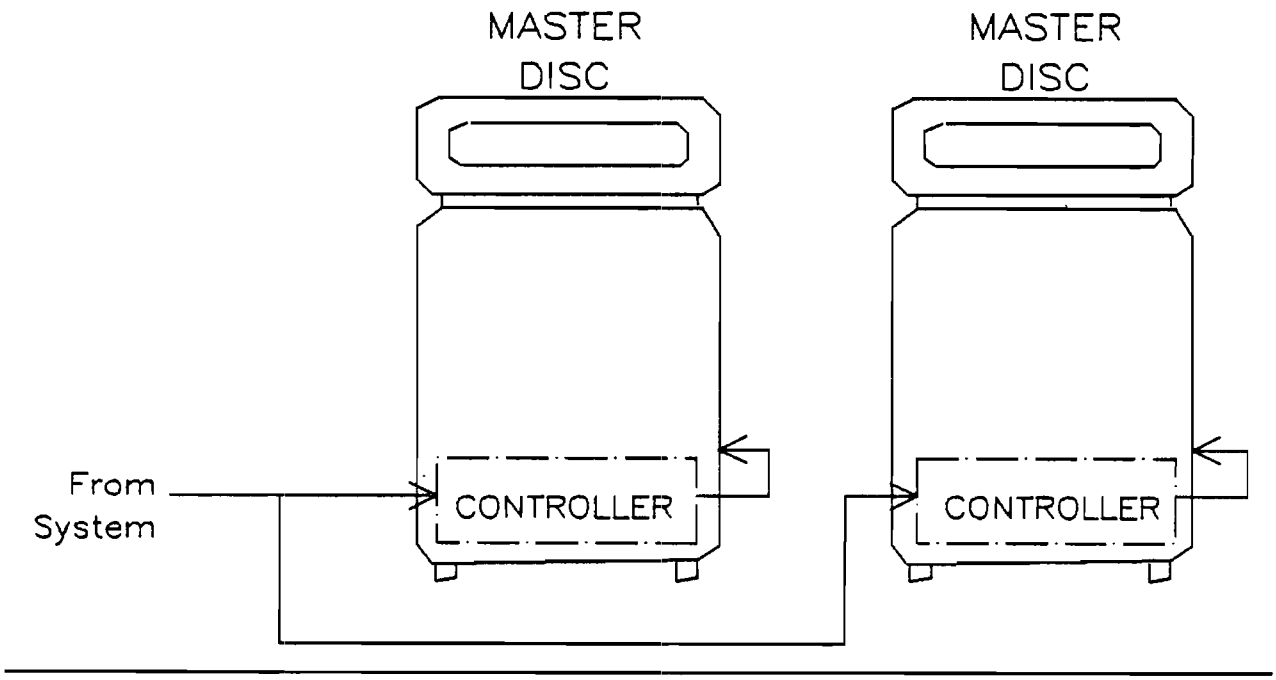


WITHOUT SEEK AHEAD

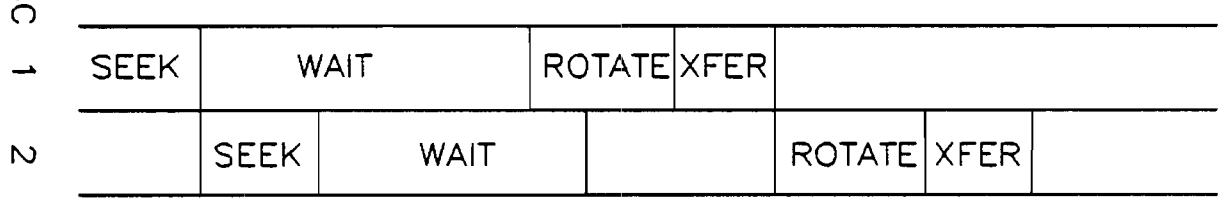


WITH SEEK AHEAD

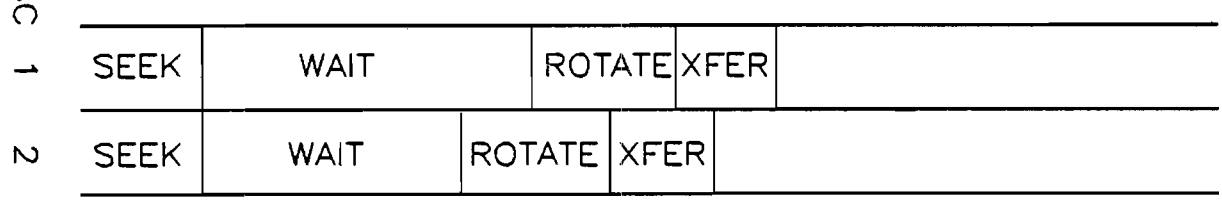
MULTIPLE MASTER DISCS



OVERLAPPED TRANSFERS

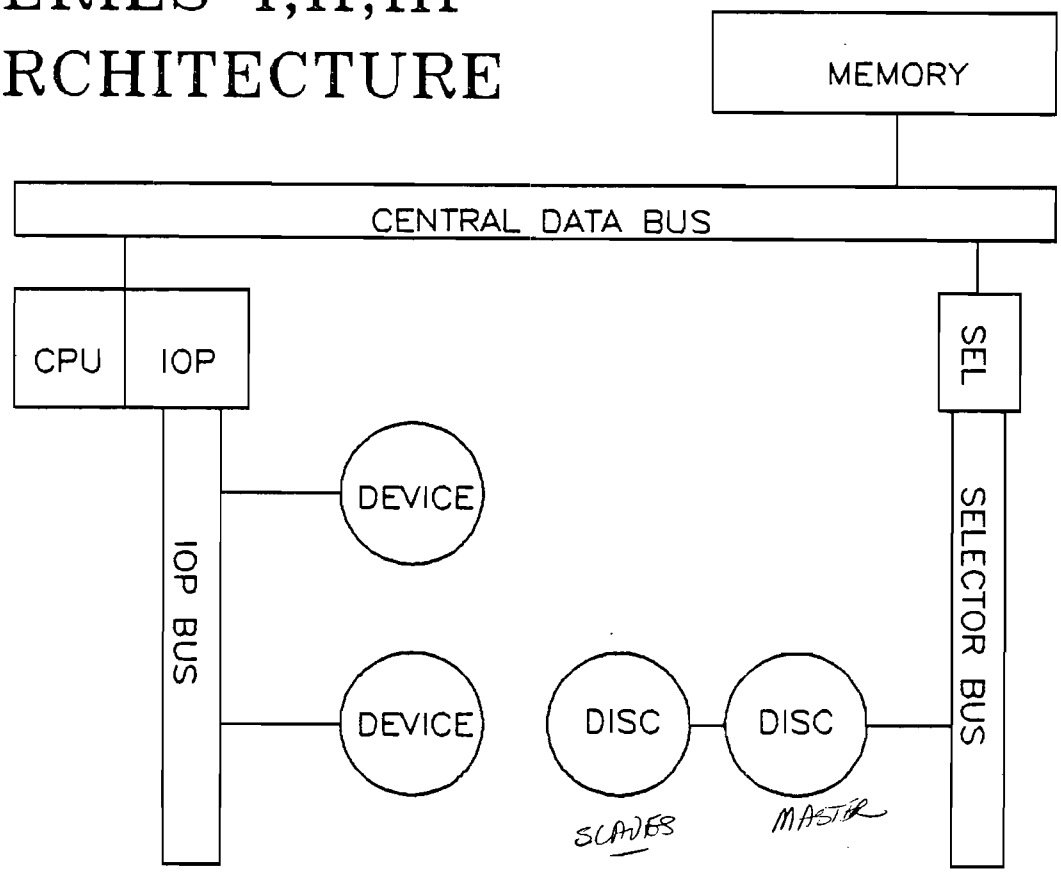


WITH SEEK AHEAD



OVERLAPPED TRANSFER

SERIES I,II,III ARCHITECTURE



DISCUSSION

The original architecture of the series I,II,III systems had the following bus speeds (approx).

MULTIPLEXER channel (IOP)	0.952	MBytes / sec outbound
	1.038	MBytes / sec inbound
Selector Channel	1.9	MBytes / sec
Central data bus	2.3	MBytes / sec

Since the Central Data Bus was capable of being saturated with data if the Multiplexor and Selector Channels were both transfering to capacity then this architecture was not expandable.

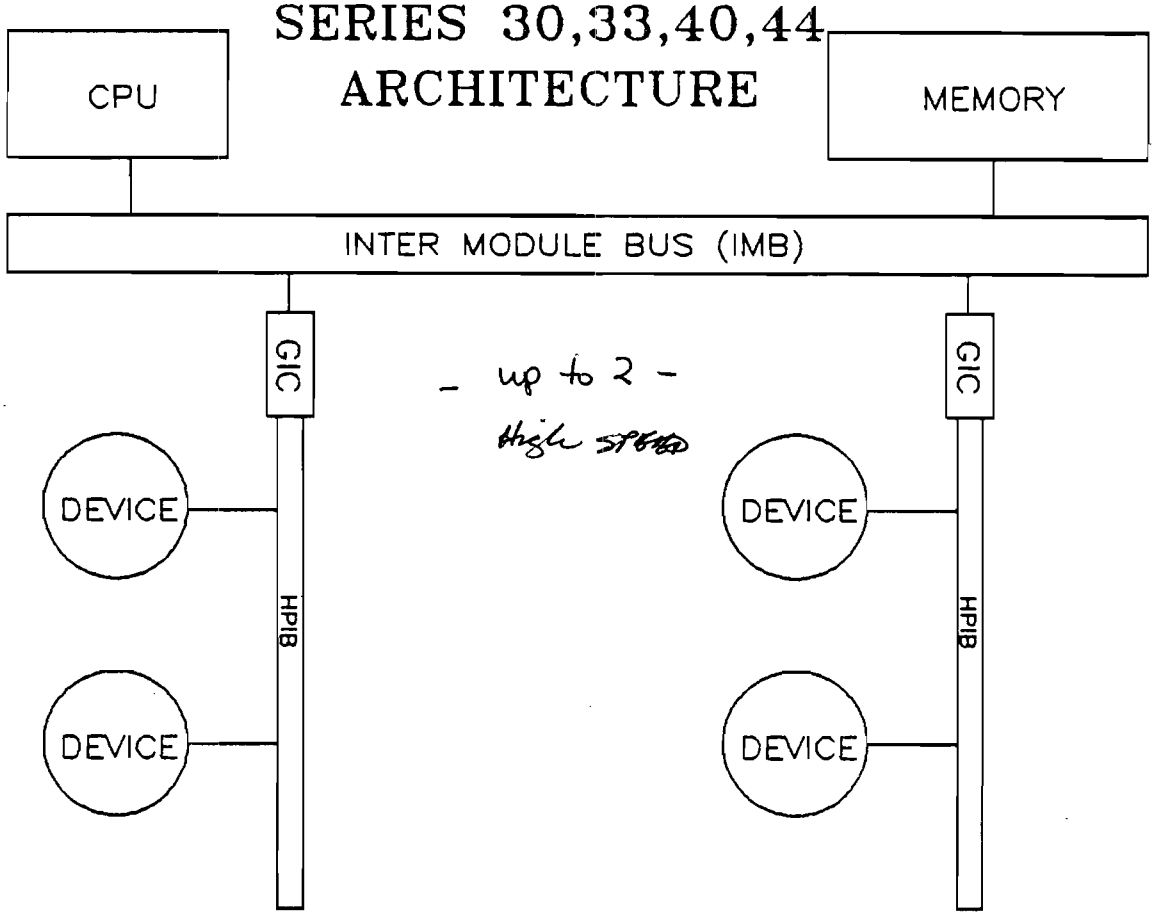
The Multiplexer channel could handle up to 16 device controllers at a time with data being transferred to all controllers at once (Word multiplexed).

The Selector channel was designed to handle high speed devices but could only handle one device controller at a time. For this reason only ONE master disc was allowed on this architecture.

NOTE: This system was not tolerant to attempts to overrun any of its data busses. If a bus was overrun then the results were unpredictable and often caused system crashes. The only solution was to configure the systems such that they could never overrun any bus.

(One passing KUDO: The series III, properly configured, was and still is one of the strongest and most reliable commercial computer systems ever built).

SERIES 30,33,40,44 ARCHITECTURE



DISCUSSION

This is the basic HPIB architecture. It form the basis for ALL the HPIB systems. Point out to the students that the series 39,40SX, 42, and 48's also use this architecture.

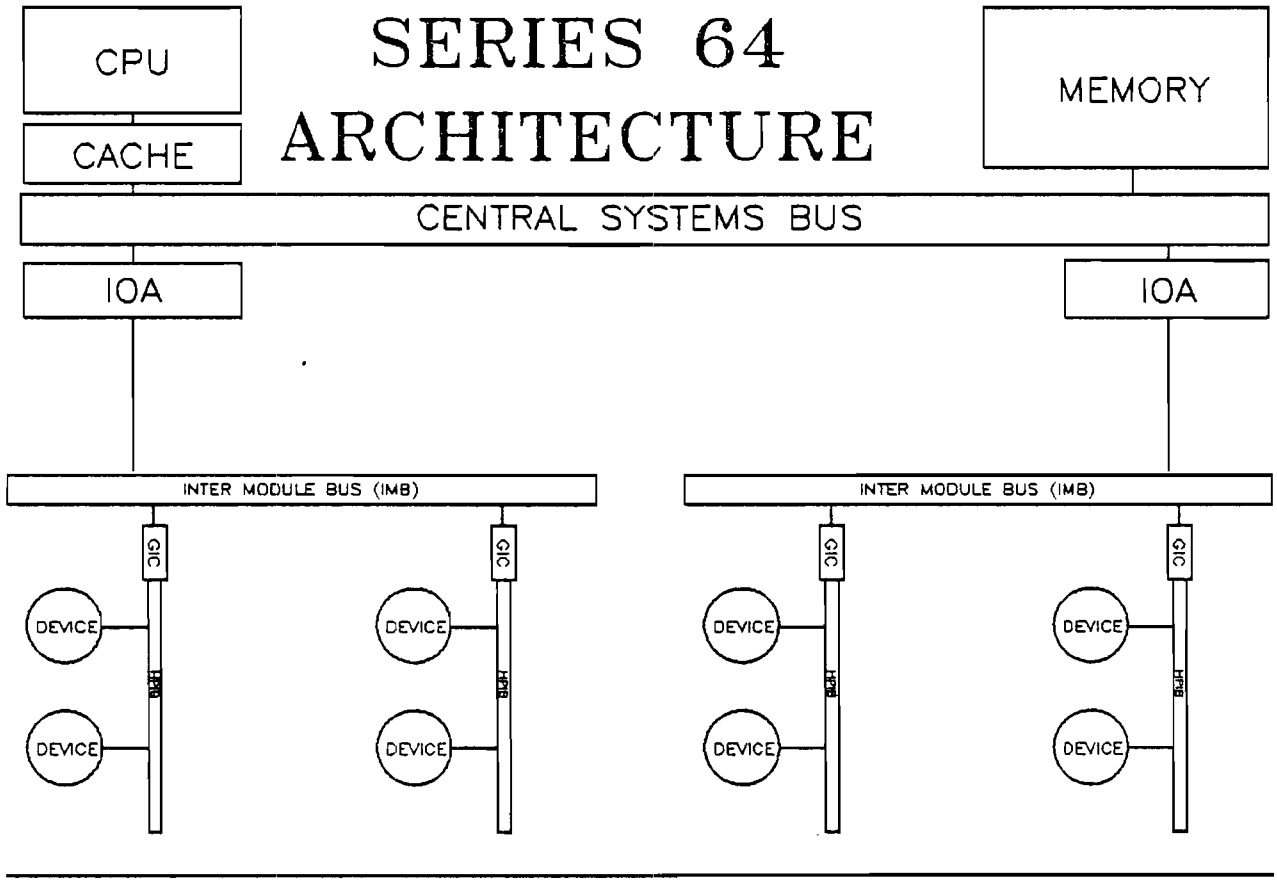
Data transfer rates for the HPIB systems are:

GIC (and HPIB itself)	1.2 MBytes / sec
IMB	3.0 MBytes / sec

One difference between this and the non HPIB systems is that overrunning the HPIB bus is NOT A PROBLEM. Intrinsic to HPIB is a method to handle requests to transfer more data than the bus can handle. The only impact of overloading the bus is that some devices will have to wait to transfer their data. As a design feature of all HPIB interfaces, this should be no problem.

The results of overrunning the IMB itself are unclear at this time. For this reason care should be taken to configure the systems so that overrunning the IBM won't occur. Actually this is not as hard as it might seem. Since most IO devices can't transfer data anywhere the maximum rate of even the HPIB then all we have to do is to control the "high speed devices".

High speed devices are normally disc drives, although there are a few others (the HP2680S, for instance, has a large buffer in it which it fills with very fast transfers from the HP3000. This qualifies it as a high speed device). If we restrict the high speed devices to be connected to only two gics max then the maximum load on the IBM can't go over 2.4 MBytes / sec. This still leaves 600 KBytes per second for all the other devices (more than enough). Since overrunning the HPIB causes no problems then any configuration where no more than two GICS have high speed devices should work properly.



64 2 IMB
68 3 IMB

DISCUSSION

The series 64 uses THE EXACT SAME IMB as the other HPIB systems. Note that the bottom left and bottom right parts of this slide are IDENTICAL to the bottom of the previous slide.

The major differences in the series 64 architecture is that it adds an IO Adapter and Central Systems Bus between the IMB and main memory. The speed of the Central Systems Bus is 56 MBytes per second (Thats right, 56 MILLION BYTES PER SECOND) so chances of overrunning it are slim. It is a wide bus (32 bit) with a transfer to memory always consisting of four 32 bit words (128 bits). The difference in the way data is actually transferred to main memory is handled by the IO Adapter. It also has a 1024 byte buffer built into it to smooth out transfers.

Actually, saying the CSB can transfer 56 MBytes / sec is a little misleading. In fact it can transfer that fast, but the main memory can't take it that fast. Main memory can transfer data at 'only' 18 MBytes / second. This is the limiting factor on how much data we can pump thru the series 64. (At least until we come up with a faster memory module).

Even so, let's do some calculations:

If an IMB can transfer a maximum of 3 MBytes / second and the CSB and main memory can transfer at 18 MBytes / second then how many IMBs could a series 64 THEORITICALLY support ?

(18/3=6 with a little fudging so the CPU can still get to main memory still allows 5 IMB's !!)

Don't try to buy a 5 IMB 64, nobody is marketing it yet. This is just to point out the growth potential of this architecture.

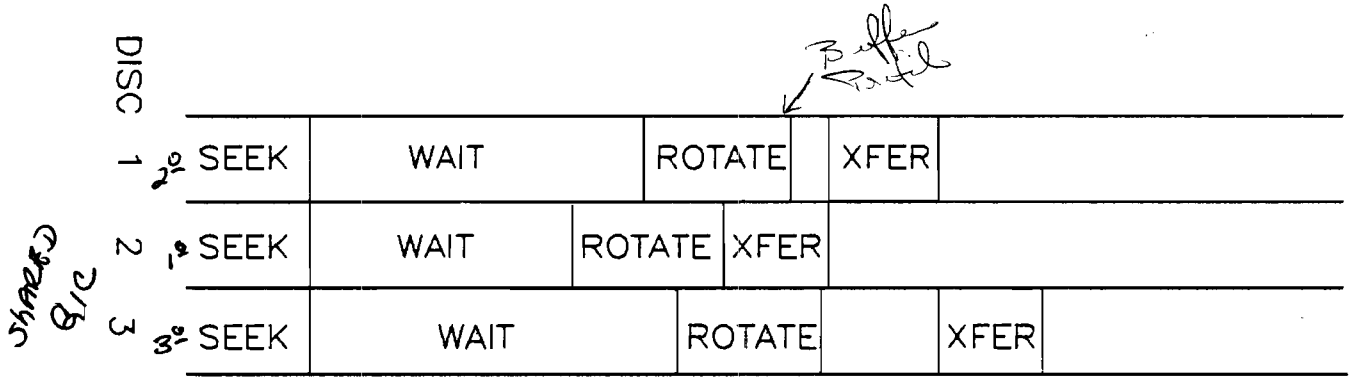
The series 64 comes standard with one IMB. You can add a second one if desired. On the series 68 (same system) you can add a third IMB. If you put two high speed GICS on each IMB you could have a total of SIX HIGH SPEED GICS on the series 68 with no possibility of overrunning any busses. Now figure 2-6 Master disc drives on each high speed GIC and you've got a considerable ability to do IO.

MAIN MEMORY CACHE

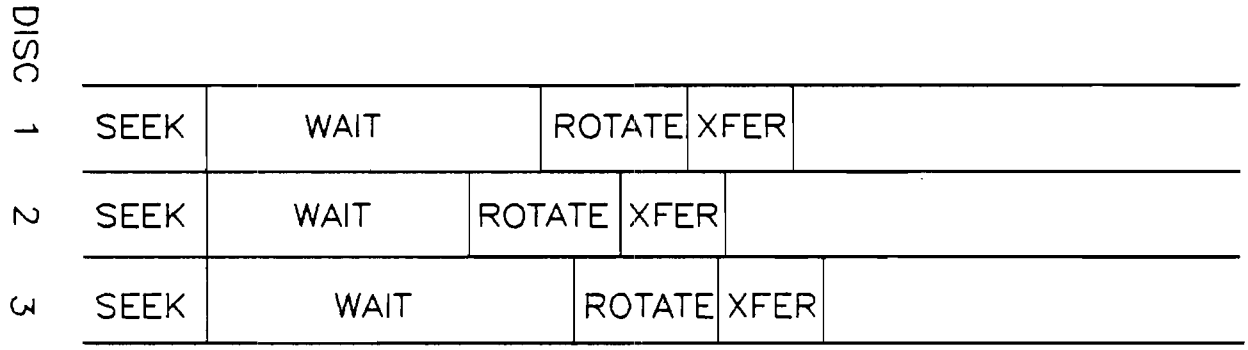
And now a word about the CACHE sitting between the CPU and the Central System's Bus. It is a very fast (75 nanosecond or 13 BILLION words per second) memory than can hold 8 KBytes of data. It is used as a buffer between the CPU and main memory with its task being to keep the most often accessed parts of main memory in itself so the CPU can access them quickly.

The fact that the cache can succeed in having what the CPU needs in memory present 95% of the time gives an effective access speed of 145 nanoseconds (13 MBytes / second). This helps the series 64 & 68 high speed CPUs to keep busy.

4KB



MULTIPLE MASTER DISCS, SINGLE GIC



MULTIPLE GICS

How long does a 7933 need the channel for?

Exclusive

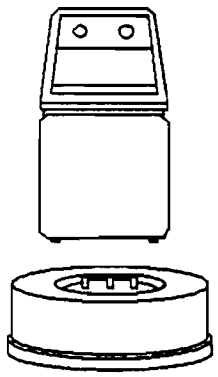
1 ms Command xfer

11 ms Latency (1 ms) rotational positional sensing

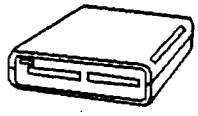
1 ms data transfer

711 I/O's sec w/RPS

BACKUP DEVICES

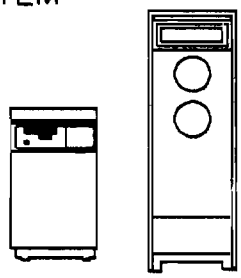


BACKUP DEVICE	MINUTES PER MB	MEDIA CAPACITY	SWITCH TIME	COST PER MEDIA
1600BPI TAPE	0.25	42MB	3 MIN	$25.00
6250BPI TAPE	0.05	156MB	1 MIN	$25.00
ICT CARTRIDGE	0.50	67MB	5 MIN	$36.00
FLOPPY	0.80	1.2MB	1 MIN	$9.40
120MB DISC*	0.02	120MB	2 MIN	$905.00
404MB DISC*	0.02	404MB	2 MIN	$1531.00



* TIMES ARE FOR DISC-TO-DISC COPY, NOT SYSDUMP

DOING BACKUP ON A 400MB SYSTEM



DEVICE	TIME	MEDIA COST
1600 TAPE	2HR 10 MIN	$250.00
6250 TAPE	23 MINUTES	$75.00
ICT	3HR 50 MIN	$216.00
FLOPPY	10HR 53 MIN	$3140.00
120MB DISC*	15 MINUTES	$3620.00
404MB DISC*	12 MINUTES	$1531.00

1978 - Streams Backup with only 1 user OPR.SYS

1974 System -

IO LAB

RUN OPT.PUB.SYS

1) on the GLOBAL screen. Can you tell if you are DISC IO Bound ? Hint = much P

   Memory Bound ? GARBAGE collected memory used.

   HOW ?

2) Go to the Disc Activity Display **(#ID)**

3) How well are the IOs spread between the master discs ?

   Evenly
   #3 maxed

4) Is one disc saturated when the others are not ? ~~no~~ yes #3

   What will this do to you maximum IOs per second for this system ?

   if saturation low

5) How can you tell if the applications are demanding more IOs per second than the system can deliver ?

   Queue length

6) What type of operations are classified as "CONTROL" ?

   FOPEN

HOW MANY DISC IOS SHOULD YOU SEE ?

1) You can't see more than the applications are demanding. (Watch out for low transaction rates and other bottlenecks such as CPU, File Locking, Image, etc.).

2) Assume the "Average" applies for the types of IO requests (Average seek, Average length transfer, etc.) but is this valid for your situation ? Consider file placement, diversity of application etc.

3) Calculate the MAXIMUM IO rate for this disc configuration. Consider masters & slaves, Disc types & characteristics, Channel loading etc. Adjust this value based on the answers to 1 & 2 above.

What you end up with is a general ball park idea of how many disc IOs per second you should expect. If your system isn't performing close to this figure then you have made a mistake in 1,2, or 3. When you find this mistake then you may have also found your performance problem.

The number of I/Os per second that a disc can sustain depends on the application request rate, the system sofware path, the controller delay and the drive characteristics.

I/O Time = Software Time + Controller Time + Disc Time

Let's take the example of a 1k byte FWRITE call on a Series 44 to a 7933:

I/O Time		Software		Controller		Seek		Latency		Transfer
36.3 ms	=	9.3 ms	+	3.8 ms	+	11 ms	+	11.1 ms	+	1.1 ms

If we convert this to an I/O rate for a stream of serialized requests:

1 / 36.3ms = 27.5 I/Os per second.

This does not account for application delay time so it may be considered an upper limit. It also assumes 50% locality. The actual locality may be higher or lower. There also may be multiple processes that are not serialized. This overlaps software execution with disc processing and can cause an increase in I/O throughput.

HOW TO PROGRAM TO DEMAND LESS IO

1) USE PROPER FILE BUFFERING

2) USE EFFICIENT PROGRAM LOGIC
   (DON'T REREAD RECORDS UNNECESSARILY)

3) USE MULTI-RECORD IO WHEN POSSIBLE
   (BEWARE OF POTENTIAL MEMORY SHORTAGES)

4) USE DEFERRED BUFFER POSTING WHEN IT MAKES SENSE (IMAGE)

5) OPTIMIZE DATA FOR THE PROGRAMS
   (PRE SORT DATA FOR KSAM & IMAGE LOADS)

6) AVOID UNNECESSARY SYSTEM OVERHEAD
   (USE FREAD/FWRITE INSTEAD OF FORMATTED IO)

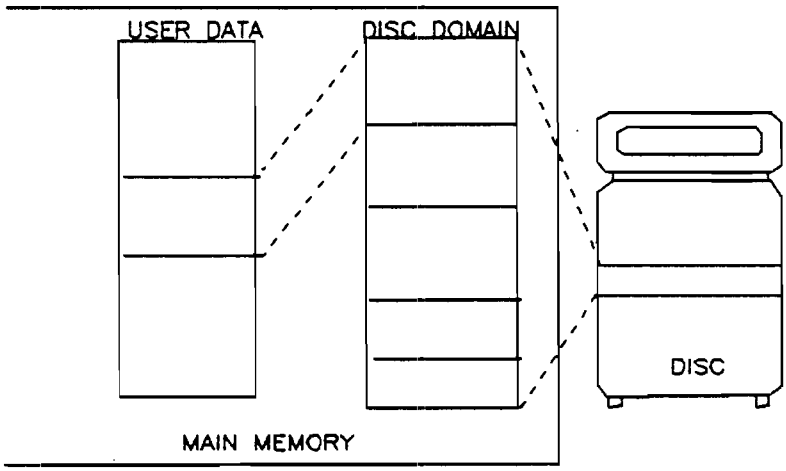
7) PROGRAM DEVELOPMENT: USE EDITORS THAT ACT DIRECTLY ON
   THE PROGRAM FILES. USE MAINTENTNCE FILE CONCEPT.

8) ANY OTHER IDEAS ?

DISC CACHE

What is it ?

It is a way to trade off excess CPU and memory in order to gain increased disc IO performance.



How does it work ?

Whenever a user requests a block of data from the disc, the system will bring in "N" times the requested data. This data is stored in a "DISC DOMAIN". The data requested by the user is then transferred to the users area. If anybody requests data that is already in a disc domain (cached) then it is transferred with no need to do an IO.

Disc caching manages retrieval and replacement of cached disc "domains" in excess main memory. Disc domains are copies of portions of disc that are maintained in main memory. No main memory is permanently dedicated to disc caching. Cached domains share main memory with all other type of MPE segments and are not treated differently by the memory manager.

Disc caching locates and replaces these cached disc domains so that a significant portion of the references to disc storage can be resolved without actually having to physically access the disc.

The disc caching policies are integrated into the MPE kernal, file system, and I/O system which allows the system performance to be tuned based on the current workload and resource availablity.

4X

Q Will disc cache increase the number of ios you can do to disc ?

A Not really. It will reduce the number of IOs requested by satisfying many directly from the disc domains. The "percieved" IO rate will go up but the ACTUAL IO rate will probably stay the same or even go down.

Q How much will disc caching help ?

A It depends: First off realize that you aren't getting something for nothing. You are trading CPU and memory usage against disc IOs. If your system is short on either CPU or memory then you might actually LOSE performance.

Next, it depends on how often the data requested is already in a disc domain. This will depend on several factors;

- The amount of excess memory on the system
- The degree to which the system is bottlenecked on IO to start with.
- The "locality" of disc accesses.
- The ratio of reads to writes on disc.

RULES OF THUMB:

1) If the CPU is busy more than 70% of the time without caching, then caching will not help. 4X type of machine. 68- 75%/80° busy –

2) If the memory management I/O rate is greater than 10/sec with caching, then caching should not be used.

Read/Write Ratio

4 : 1 Optimum

R < W Forget cache

Locality

50% of Mem

SYSTEM EFFECTS ON DISC PERFORMANCE

Earlier we saw that a single disc is capable of 27 I/Os per second. Why is it then tat the I/O rate actually measured on the system is rarely that high?

First let's distinguish between response time [performance] and throughput [I/O rate]. The average disc service time is about 36 ms per I/O. The smaller we can make that time, the better system performance will be. The inverse of that number, 27.5 I/Os per second, suggests how many I/Os could be performed if the disc were kept 100% busy.

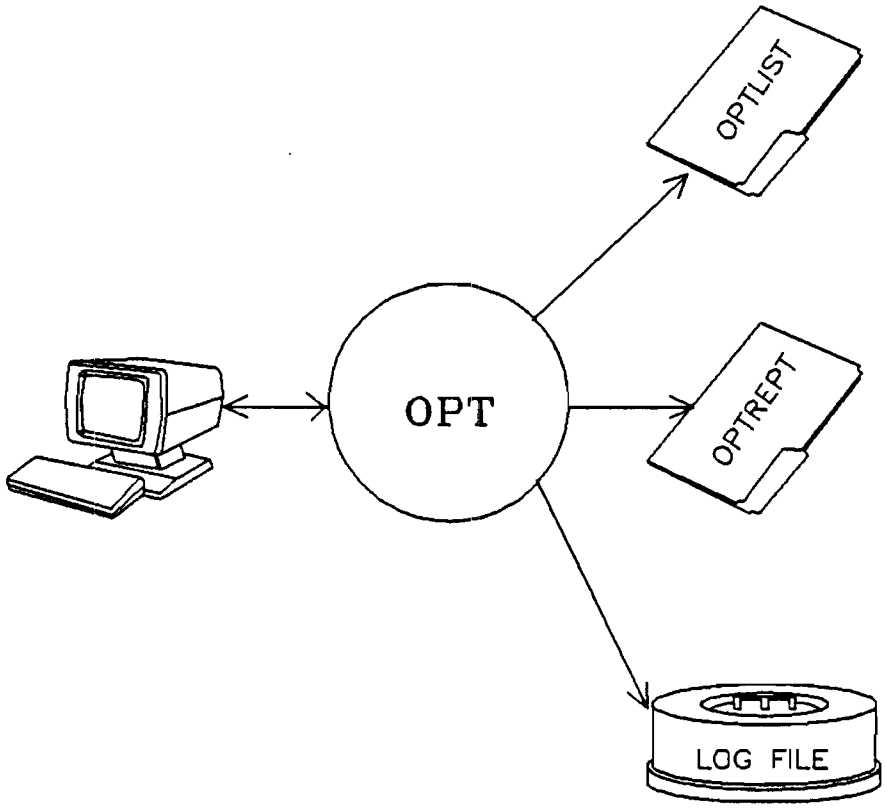
Having a 100% busy disc is a rarity for a variety of reasons: There may be few processes in the system demanding I/O. The processes that need I/O may be making most of their requests to a single drive leaving the remaining drives virtually idle. Many processes are serialized, i.e. they must pause to do some or maybe lots of processing before requesting another I/O. It may be necessary to wait for another I/O device (e.g. tape, terminal, plotter, etc.) before requesting another disc I/O. Whatever the reason, discs are often idle and the throughput rarely approaches maximum.

One of the best ways to get better system performance is to eliminate as much I/O as possible. Take the example of copying a text file. The typical text file block size is 5 sectors (16 * 80 bytes). Copying a 1000 line document involves roughly 60 disc I/Os. Increasing the block size to 15 sectors will cause the number of I/Os to go down to about 20. The performance of the system will go up while the I/O rate goes down.

Another example of the same concept is "disc caching" which stores actively used data in main memory. This eliminates I/Os to the disc by retrieving the data from main memory. With disc caching the system performance goes up because of reduced dependence on disc I/O.

In a multiple disc configuration, having a balanced load across the available discs is important. Too much demand for a single disc can cause bottlenecking on that disc while another disc may be totally idle. It can be beneficial to spread active files amount the various drives to achieve a good balance. Keep in mind that there is a natural tendency under MPE to have lots of activity on LDEV1, so that active user files, virtual memory, etc. may need to be concentrated on other discs to achieve the proper balance.

OPT LOGGING AND SUMMARY REPORTS



COMMANDS THAT GENERATE HARD COPY

(on OPTLIST)

L Generate a hard copy of the currently displayed screen. The output goes to file OPTLIST which defaults to device class LP.

The first time you issue this command you will be prompted

ENTER HARD COPY IDENTIFICATION

This ID will be printed on each hard copy.

#H Generate continuous hard copies of each screen displayed.

R (From the GLOBAL Context only)

Generate a single Summary Report on OPTLIST.

#L Close the OPTLIST file. (Release the spool file for printing).

GLOBAL SUMMARY REPORT COMMANDS

(On OPTREPT)

A (Only if issued from the GLOBAL context)

Activates periodic summary report generation. (Output goes to file OPTREPT which defaluts to device class LP). You will be asked the following questions:

MINUTES BETWEEN SUMMARY REPORTS ?

REPORTS BETWEEN CLOSING OF REPORT FILE ?

NAME OF THE LOG FILE ?

ENTER SUMMARY REPORT IDENTIFICATION
(Max of 25 Characters)

S (Must be issued from the GLOBAL context)

STOP summary report generation and close OPTREPT file.

Z Zero out the accumulating totals on the summary reports.

RUNNING OPT FROM A BATCH JOB

```
:JOB OPT,MANAGER.SYS
:RUN OPT.PUB.SYS                                    DEFAULTS

MINUTES BETWEEN SUMMARY REPORTS ?                   (60)

REPORTS BETWEEN CLOSING OF REPORT FILE ?            (8)

FILE CLOSINGS BEFORE TERMINATION ?                  (3)

NAME OF LOG FILE ?                                  (none)

SUMMARY REPORT IDENTIFICATION ?                     (none)

:EOJ
```

OTHER OPT COMMANDS

T Change the time delay between screen updates.

: Execute MPE commands

! Place a note in the OPTLIST report.

#S Suspend OPT until a key is pressed (or 15 minutes which ever comes first)

#A Alter the position of the Line Drawing character set to agree with your terminal.

X Execute a command sequence repeatedly.

Example:

```
>X
>#M MO#C  #I L#PF#TDL#GRL
```

(What would this do ?)

(How do you stop it ?)

OPT LOG FILES

The OPT LOG FILES are disc files created by OPT during the SUMMARY REPORT operation. They can be created either interactively or from a batch job. They contain the summary reports in binary form.

What can you do with these files ?

Reproduce the summary reports using the contributed program OPTREPT.

Plot system resource usage using DSG/3000.
(Requires a user written program to format the data).

Other types of analysis depending on individual requirements.

INTERPRETING THE SUMMARY REPORTS

CPU Activity Summary: (How the CPU spent its time)

Memory Allocation Summary:
(How the memory manager spent its time)

Launch Activity (How the dispatcher spent its time)
and Additional Memory Management Activity Summary
(More details on memory management)

Summary of Disc Activity:
(What was the loading and balance on the discs?)

Summary of Line Printer Activity:

Summary of Tape Activity:

OPT REPORTS LAB

1) Create a batch job that runs OPT and produces summary reports as follows: 1 report every minute and a total time of ten minutes before the job terminates. Include a summary report identification that contains your name.

2) Wait until the instructor places the system under an artificial load then stream your job.

3) Use OPT on line while the load is running. Generate a hard copy of any displays that might indicate possible bottlenecks. Use notes in the hard copy listing to remind yourself of why you generated each hard copy.

4) At the termination of the batch job retrieve your listings and prepare your best assessment as to what the condition of the system was.

INCLUDE: BOTTLENECK(S)

OFFENDING PROGRAM(S) or USER(S)

POSSIBLE ACTIONS TO IMPROVE SYSTEM PERFORMANCE

WHAT MIGHT HAPPEN IF YOUR RECOMENDATIONS ARE FOLLOWED

(HOW MUCH FASTER WOULD THE SYSTEM GO AND WHAT WOULD BE THE NEXT BOTTLENECK) ? 5) If you get done early then reflect on the next few pages in your workbook. The class as a whole will discuss and fill in these pages to characterize bottlenecks.

BOTTLENECK SYMPTOMS

SYSTEM SITUATION: MEMORY BOTTLENECKED

Slow Response time: #G 50-100% full -, some busy. D/G
Lots of Segment I/O

SYSTEM SITUATION: CPU BOTTLENECKED

Globals shows no power Table -
if you get above 80 (Busy)
Show Q: The More than one in the Dispatch Queue (Right hand)

SYSTEM SITUATION: DISC BOTTLENECKED

#G Pause for I/O

SYSTEM SITUATION: IMAGE BOTTLENECKED

Tap Processes
Dbutil shows Locks

SYSTEM SITUATION: APPLICATION LIMITED

USG & access resources.
MM/FA Bottleneck!

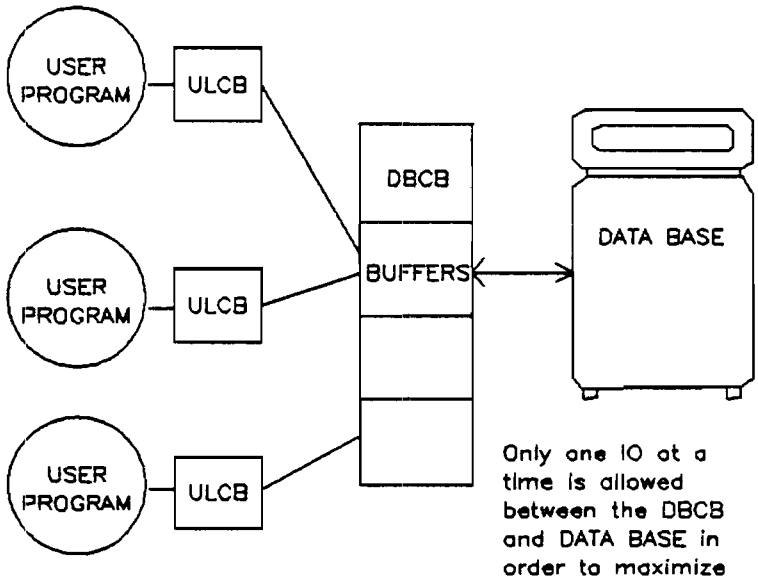
SYSTEM SITUATION: NO BOTTLENECKS

SYSTEM SITUATION:

SYSTEM SITUATION:

SYSTEM SITUATION:

THE "IMAGE" BOTTLENECK



Only one IO at a time is allowed between the DBCB and DATA BASE in order to maximize data base integerty.

ULCB = User Local Control Block

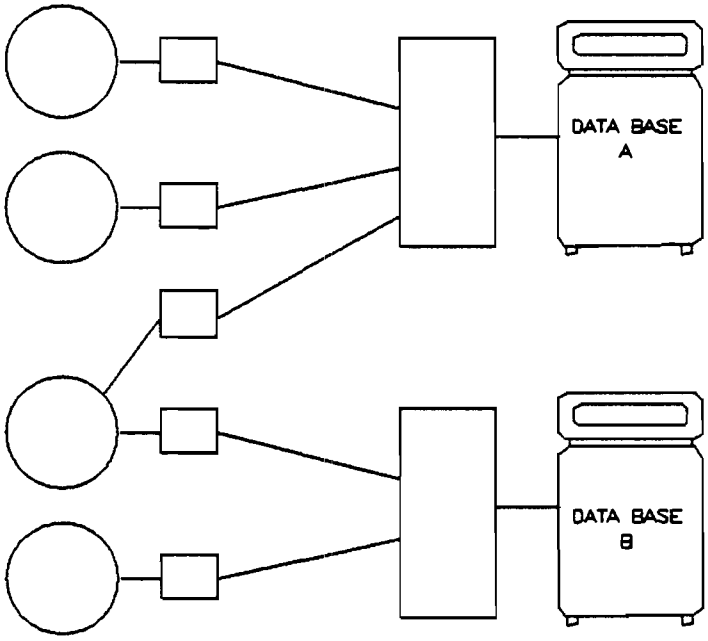
DBCB = Image's Global Data Base Control Block

How many IOs per second can you expect on a single data base ?

Would adding master disc drives help ?

Would Disc Caching help ?

MULTIPLE DATA BASES



How many IOs per second can you expect from

- Data Base A ?
- Data Base B ?

If Data Base A & B were combined into a single data base, what would the resulting performonce be ?

DATA BASE or FILE LOCKING BOTTLENECKS

A "LOCK" is applied to prevent other users from accessing data during critical operations.

Those users will be held off, or "IMPEDED" until a corresponding "UNLOCK" is performed.

NOTE: Locking is a feature that is necessary to preserve data integrity but used unwisely it can cause a system bottleneck.

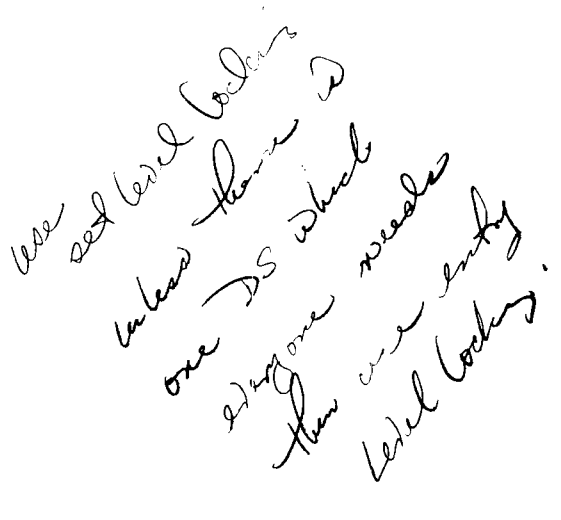
How can you tell if locking is bottlenecking your system ?

OPT will show processes being "held off" by a lock in the IMPEDED state. (Be careful as there are situations other than locking that can show up in the IMPEDED state also. DBCB queueing in IMAGE for example).

To determine if you have a data base locking problem

:RUN DBUTIL.PUB.SYS
>SHOW database LOCKS

APS/3000 can also point out locking problems. It will show a large percentage of "WAIT" time at calls to lock a file or data base.



DATA BASE LOCKS LAB

1) The instructor will start a program that uses an IMAGE data base.

2) RUN DBUTIL.PUB.SYS

3) Determine how many processes are accessing the data base.

4) Is IMAGE locking causing a bottleneck in this case ?

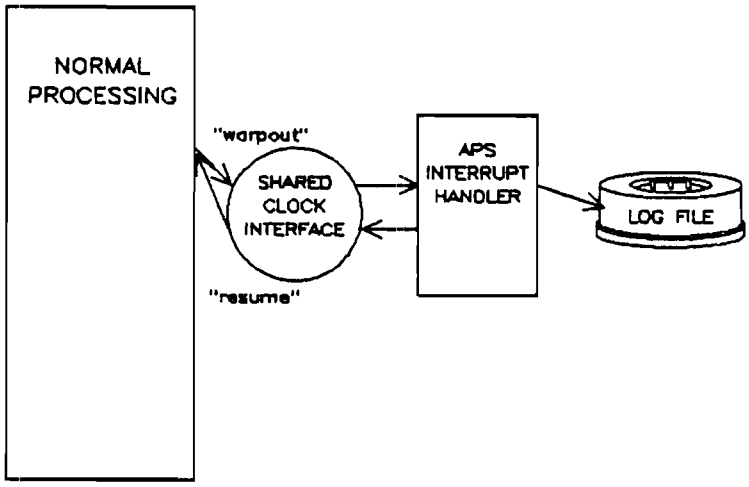
5) How can you tell ?

6) What would help this application ?

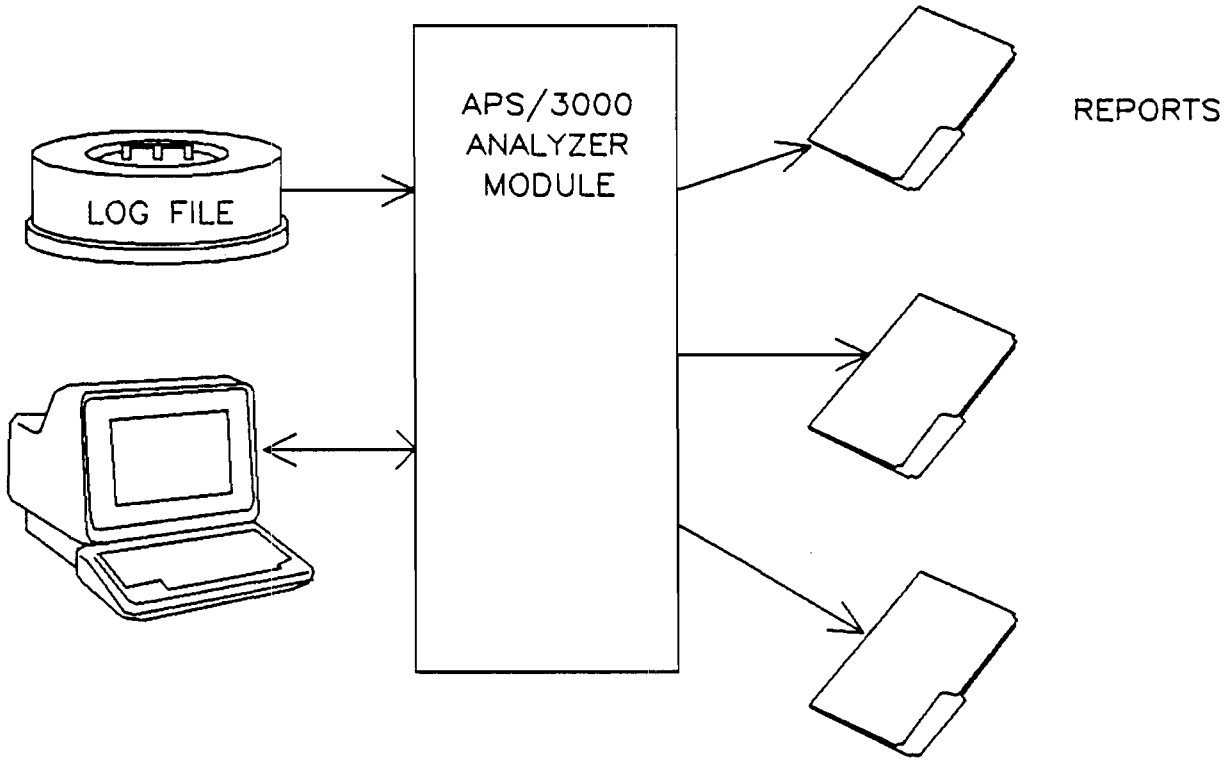
APS/3000 (SAMPLER)

This program uses the "Shared Clock Interface" facility of MPE to install its own special interrupt handler routine on the system. The system clock is then set to interrupt the CPU and execute this routine at a preset interval (default=20ms). The interrupt handler quickly determines what was going on (where the CPU was when it was interrupted) saves this information then resumes execution.

This information can be saved in a log file and analyzed in detail later. It will show histograms of where the CPU was spending its time.



The APS/3000 analyzer module can then analyze the log file and produce reports showing where it found the CPU.



The APS/3000 reports are structured into levels:

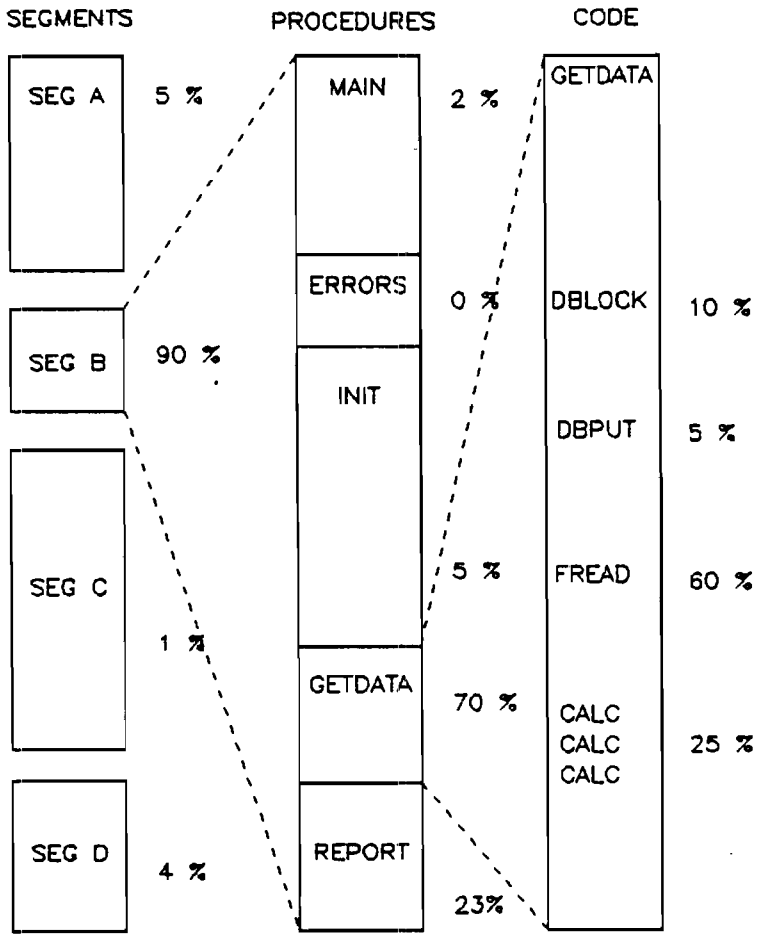
1) GLOBAL SUMMARY (How often was the CPU paused, running a user program, or running system processes)

2) PROGRAM FILE AND PROCESS LEVEL (Which programs and processes (PINS) was the CPU spending its time on ?)

3) SEGMENT LEVEL (Which code segments, system and user, was the CPU using ?)
This level can also INFER inter segment transfers and WAIT times.

4) PROCEDURE AND ADDRESS LEVEL (Where, within each segment, was the CPU time spent ? This will be by segment relative address, or, if a PMAP or FPMAP is present, by procedure and procedure relative address.)

The purpose of the APS/3000 reports is to identify that area of code where the CPU is spending the majority of its time. Often you will find that the CPU spends 80% of its time executing only 10 % of the code.
Optimizing anything but that 10% of the code will produce little overall effect on the application's performance.


SEGMENTS
PROCEDURES
CODE
SEG A
5 %
SEG B
90 %
SEG C
1 %
SEG D
4 %
MAIN
ERRORS
INIT
GETDATA
REPORT
2 %
0 %
5 %
70 %
23%
GETDATA
DBLOCK
10 %
DBPUT
5 %
FREAD
60 %
CALC
CALC
CALC
25 %

APS/3000 LAB

1) Select a program for analysis. You should have a source code listing with code locations included.

```
SPL         $CONTROL LIST
FORTRAN     $CONTROL LIST,LOCATION
COBOL       $CONTROL LIST,SOURCE,MAP
COBOL-II    $CONTROL LIST,SOURCE,VERBS
PASCAL      $CODE_OFFSETS ON$
BASIC COMP  $CONTROL LIST,SOURCE,LABEL
RPG         (SORRY, APS/3000 WILL BE OF LITTLE HELP ON RPG)
```

You should also have a PMAP available, preferably

```
:PREP $OLDPASS,PROGRAM;FPMAP
```

or acceptably

```
:PREP $OLDPASS,PROGRAM;PMAP
```

2) RUN SAMPLER.PUB.SYS (Only one user at a time will be allowed to run APS/3000 in the data gathering mode. so coordinate this activity with the other students. Any number can run APS/3000 in the data analysis mode).

3) Take a new measurement **NM**

Run & monitor a program **RM**

Begin & Log data **BL** default the sampling interval, Take Indirect CPU and WAIT time data

Execute the program.

4) After the program has completed, analyze the log file. (NOTE: other students can gather APS data at this time).

5) Display and get hard copy of the following reports:

-Direct & Indirect CPU utilization by programs.
-Direct CPU utilization by All Segments
-Indirect cost of individual user segments that had > 5% direct or indirect CPU time.
-Direct CPU time by segment relative address.
-Direct CPU time by procedure relative address.
-Direct & Indirect CPU time by procedure relative address.
-Wait times by procedure relative address.

6) Exit Sampler **EX**

7) Mark your source listing to show those areas where a high percentage of CPU time was spent. Include Direct, Indirect and Wait times.

   Indicate areas where the program's performance might be increased. What would you do in each of these areas ?

   What potential for performance increase do you estimate might be gained in each major area ? What would it cost to change it ?

8) Further analysis of the log file can be done by RUNNING SAMPLER again and REPLAYing the log file **RE**. Try it.

9) When you are finished, purge your log file.

10) Were you suprised by what you found ?

MPE "HOOKS" FOR GATHERING PERFORMANCE DATA

* MON

Original method, copies minute events of system activity and logs then to a dedicated mag tape drive. It can add 50% overhead during the measurement. Analysis time of the data is substantial.

* Measurement Interface Facility

Accumulates STATISTICS of system performance "realtime" and makes them accessable to on-line tools like OPT/3000.
Overhead added is negligable and analysis is done "real time".

* Shared Clock Interface

"Snapshots" the system via an interrupt handler sharing the system clock interrupts. Low overhead unless sampling rate is set too high. Used by APS/3000

* MPE Log files

Low resolution events are logged to disc files. They can be analyzed to provide longer interval analysis of system performance. Overhead is minimal so this tool can be left on at all times.

OTHER SOURCES FOR PERFORMANCE INFORMATION

SYSTEM CONFIGURATION
- SYSDUMP
- SYSINFO
- TUNER

LOADING
- SHOWJOB
- SHOWQ
- DBUTIL (SHOW USERS)
- MPE LOG FILES & UTILITIES
- SURVEYOR
- SOO

OTHER TOOLS
- APS/3000
- HP TOOLS (DCP, C&I, TEPE)

How about including performance measurement "hooks" into the user application programs. They could measure transaction rate, response time, CPU time uses, types of transactions etc. to a user log file.

OTHER TOOLS LAB

Run or examine output from any other performance analysis tools depending on their availablity. Instructor's option.

WHAT IF THERE ARE NO SYSTEM BOTTLENECKS ?

Perfect ! Response time is instantaneous, right ?
Wrong ! Each transaction will still take some minimal amount of time to complete, even if the system has no bottlenecks.

FOR EXAMPLE:

1)A user enters data then presses "return".

2)The application performs the transaction
- consumes 1 second of CPU time (1 sec)
- Does 120 Disc IOs (4 sec)

3) Application returns information to user terminal (2 sec)

4) Ready for new data

The minimum time for this transaction to complete is seven seconds. The only way to make it go faster is to either

A: make the system faster (faster CPU, change DISC IO configuration, etc).
or B: Make the application demand less of the system in order to perform this transaction

IMPROVING SYSTEM PERFORMANCE

Step 1) Apply the system load under Light, Average, and Heavy Activity

2) Measure the system's response at each level..

3) Determine what the system bottlenecks are.

4) Decide on a course of action
- Reduce the application's demand on critical resources.
- Increase the system's ability to provide those resources.
- Trade off critical for non-critical resource usage.

CAUTION: Murphy's Law of Bottlenecks states

"There is ALWAYS one more bottleneck".

Watch out for resources that are heavily used but not bottlenecking the system. If you remove the current bottleneck then these resources will become your next bottleneck. Don't trade off one bottleneck for another one that will be just as restrictive.

IMPROVING PERFORMANCE LAB

1) The instructor will impose a controlled load on the system.

2) Measure the system response using whatever tools you feel are appropriate.

3) What are the bottlenecks ?

   What are the potential bottlenecks ?

4) What would you do to improve performance ?

5) Instructor's option to implement your suggestions and re-run the load to test your theories.

FORECASTING SYSTEM PERFORMANCE

1) Determine what applications and transactions will be running on the system.

Include the number of simultaneous users and the average transaction rate of each one.

2) Measure the use of the system resources by each transaction type. (CPU, IO, MAIN MEMORY etc.).

Run the transaction on a stand–alone system and measure overall usage

or

Include resource measures into the program whenever possible

or

Measure the overall system under actual load then use the number & type of transactions processed to derive the resources consumed by each one.

4) Compare these figures against the system's ability to provide each resource.

i.e. if > 100% CPU time will be required then performance will be degraded by CPU speed.

if more IOs per second will be demanded than the system can provide then performance will be degraded by DISC IO.

If more memory will be required than is available then the system will be memory bound. (and even shorter of disc IOs).

Don't forget to include MPE's use of each resource. (For instance, the Memory Manager can require substantial disc IOs if the system is memory bound).

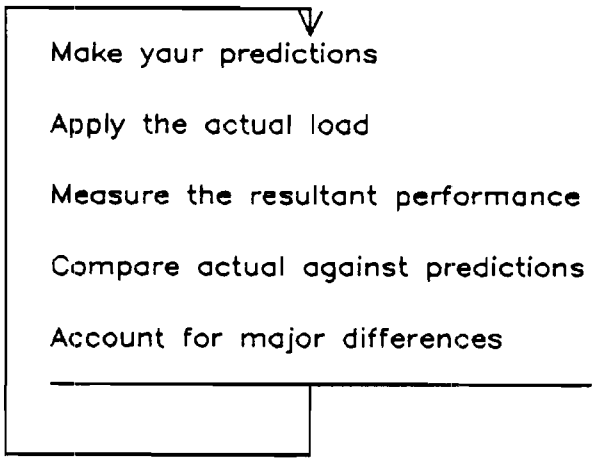
5) Lastly consider Queueing mechanisms in effect such as the IMAGE DBCB, File and Data Base locking etc.

Now apply all your knowledge of how the system performs under load and come up with an "educated guess" at the resultant performance.

(Yes, performance analysis is still more of an art than a science)

The more experienced you become, the better "guess" you can make.

the way to become experienced is thru a positiive feedback loop:



FORECASTING LAB

1) The instructor will describe the load to be imposed on the system.

2) A light load will then be imposed so you can measure its use of system resources. Use any appropriate tools.

3) Forecast system performance when placed under desired load:

   - How many users can be run with acceptable performance, given the current system configuration ?

   - What will be the first bottleneck ?

   - What can be done to allow the desired load to perform properly ?

4) The instructor will then apply the desired load on the system.

5) Measure the actual performance and compare it to your forecast. Explain any major differences.

6) RE-forecast what can be done to improve performance if necessary.

7) Instructor's option to effect changes and re-run the desired load for confirmation.

STEPS TO MAKING A SUCESSFUL PERFORMANCE ANALYSIS

1)Determine the objective
- -- Improve current performance
- -- Forecast future performance

2) Characterize the system load
- -- Interactive / batch
- -- # and type of transactions, rates
- -- Talk to the end users **

3) Select the tools you will use
- -- OPT
- -- Log files
- -- APS/3000
- -- Application "hooks"
- -- Scripting
- -- others

4) Collect data
Make sure the system load is what you want to measure or the data will be useless.

** Observe the end user interactions.

5) Analyze the data
   - Compare anticipated load against actual
   - Find current & potential bottlenecks
   - Identify any excess resources

6) Select a course of action

7) Present your results and suggestions to others in a fashion that can be understood without in-depth knowledge. (Eliminate JARGON).

8) Perform the course of action

9) Measure the system performance again.

SUPER LAB

(Optional as facilities are available)

Actual system performance under real-life loads can be measured if access to a production system is available. If not then log files and reports taken from such a system may be used.

Using all the skills at your disposal, evaluate the system performance and make recomendations as to bottlenecks, potential bottlenecks, the systm's ability to handle increased load, and actions that might be taken to either improve current performance or enable it to handle increased loads.

Present your findings such that they can be understood by the end users.

```
** END OF FORMATTING **
TDP/3000    (A.03.08)  HP36578 Formatter
THU, FEB  2, 1984, 11:47 AM
NO ERRORS
INPUT  = K0331055.SLIDES.PERFCLAS
OUTPUT = *HP2680
```

TEST 1

INTERVAL LENGTH: 140.032 SECONDS (**2.3 MINUTES**)

CPU ACTIVITY SUMMARY

CPU STATE	MEAN	MAX	LENGTH	COUNT	TOTAL TIME
CPU BUSY	98%	99%	.218	633	137.762
PAUSE DISC & SWAP	0%	0%		0	.000
PAUSE DISC	0%	0%		0	.000
PAUSE SWAP	0%	0%		0	.000
PAUSE IDLE	0%	0%		0	.000
GARBAGE COLLECTION	0%	0%		0	.000
MEMORY ALLOCATION	0%	0%	.010	6	.062
ICS OVERHEAD	2%				2.208

MEMORY ALLOCATION SUMMARY

RESULT	MEAN	COUNT
RECOVERY	25%	2
FREE SPACE	37%	3
OVERLAY CAND	38%	3
GIVE UP	0%	0
HARD REQUEST	0%	0

LAUNCH ACTIVITY AND ADDITIONAL MEMORY MANAGEMENT ACTIVITY SUMMARY

	PROCESS LAUNCHES	PROCESS SWAP-INS	PROCESS PREEMPTS	MEMORY ALLOCS	SPECIAL REQUESTS	MM I/O READS	MM I/O WRITES	RELEASE DATA SEG	RELEASE CODE SEG	CLOCK CYCLES
COUNT	633	6	94	8	0	6	3	1	1	1
RATE	4.5	.0	.7	.1	.0	.0	.0	.0	.0	.0
MAX RATE	6	0	2	0	0	0	0	0	0	0

SUMMARY OF DISC ACTIVITY

COUNT/RATE

MAXIMUM RATE (USER/MM)

	ALL I/O	READS	WRITES	CONTROL OPS	READS	WRITES	CONTROL OP
ALL DISC	40/ .3	19/ .1	17/ .1	4/ .0	1/ 0	0/ 2	0
DISC 1 (LDEV 1)	23/ .2	16/ .1	5/ .0	2/ .0	1/ 0	0/ 2	0
DISC 2 (LDEV 2)	3/ .0	1/ .0	1/ .0	1/ .0	0/ 0	0/ 0	0
DISC 3 (LDEV 3)	14/ .1	2/ .0	11/ .1	1/ .0	0/ 0	0/ 0	0

SUMMARY OF LP ACTIVITY

COUNT/RATE

MAXIMUM RATE

	ALL I/O	WRITES	CONTROL OPS	WRITES	CONTROL OP
ALL LP	0/ .0	0/ .0	0/ .0	0	0

SUMMARY OF TAPE ACTIVITY

COUNT/RATE

MAXIMUM RATE

	ALL I/O	READS	WRITES	CONTROL OPS	READS	WRITES	CONTROL OP
ALL TAPE	0/ .0	0/ .0	0/ .0	0/ .0	0	0	0

what memory what CPU

TEST 2

INTERVAL LENGTH: 67.328 SECONDS (1.1 MINUTES)

CPU ACTIVITY SUMMARY

CPU STATE	MEAN	MAX	LENGTH	COUNT	TOTAL TIME
CPU BUSY	6%	6%	.004	1066	3.836
PAUSE DISC & SWAP	6%	6%	.046	95	4.344
PAUSE DISC	1%	1%	.013	29	.388
PAUSE SWAP	50%	52%	.048	729	34.776
PAUSE IDLE	2%	2%		0	1.070
GARBAGE COLLECTION	11%	11%	.007	973	7.093
MEMORY ALLOCATION	12%	12%	.004	2151	7.781
ICS OVERHEAD	12%				8.040

MEMORY ALLOCATION SUMMARY

RESULT	MEAN	COUNT
RECOVERY	16%	373
FREE SPACE	19%	439
OVERLAY CAND	14%	315
GIVE UP	44%	979
HARD REQUEST	7%	169

LAUNCH ACTIVITY AND ADDITIONAL MEMORY MANAGEMENT ACTIVITY SUMMARY

	PROCESS LAUNCHES	PROCESS SWAP-INS	PROCESS PREEMPTS	MEMORY ALLOCS	SPECIAL REQUESTS	MM I/O READS	MM I/O WRITES	RELEASE DATA SEG	RELEASE CODE SEG	CLOCK CYCLES
COUNT	1066	2151	9	2442	7	751	439	397	338	98
RATE	15.8	31.9	.1	36.3	.1	11.2	6.5	5.9	5.0	1.5
MAX RATE	16	32	0	36	0	11	7	6	5	1

SUMMARY OF DISC ACTIVITY

COUNT/RATE

	ALL I/O	READS	WRITES	CONTROL OPS
ALL DISC	1317/ 19.6	797/ 11.8	459/ 6.8	61/ .9
DISC 1 (LDEV 1)	525/ 7.8	327/ 4.9	179/ 2.7	19/ .3
DISC 2 (LDEV 2)	268/ 4.0	124/ 1.8	128/ 1.9	16/ .2
DISC 3 (LDEV 3)	524/ 7.8	346/ 5.1	152/ 2.3	26/ .4

MAXIMUM RATE (USER/MM)

READS	WRITES	CONTROL OP
1/11	0/15	1
0/ 4	0/ 6	0
0/ 2	0/ 4	0
0/ 5	0/ 5	0

SUMMARY OF LP ACTIVITY

COUNT/RATE

	ALL I/O	WRITES	CONTROL OPS
ALL LP	0/ .0	0/ .0	0/ .0

MAXIMUM RATE

WRITES	CONTROL OP
0	0

SUMMARY OF TAPE ACTIVITY

COUNT/RATE

	ALL I/O	READS	WRITES	CONTROL OPS
ALL TAPE	0/ .0	0/ .0	0/ .0	0/ .0

MAXIMUM RATE

READS	WRITES	CONTROL OP
0	0	0

NO memory.

TEST 3

INTERVAL LENGTH: 64.617 SECONDS (1.1 MINUTES)

CPU ACTIVITY SUMMARY

CPU STATE	MEAN	MAX	LENGTH	COUNT	TOTAL TIME
CPU BUSY	39%	42%	.005	4664	24.749
PAUSE DISC & SWAP	5%	15%	.020	160	3.191
PAUSE DISC	28%	36%	.014	1314	18.383
PAUSE SWAP	0%	0%		0	.000
PAUSE IDLE	0%	2%	.317	1	.317
GARBAGE COLLECTION	1%	3%	.009	80	.740
MEMORY ALLOCATION	2%	6%	.010	117	1.192
ICS OVERHEAD	25%				16.045

MEMORY ALLOCATION SUMMARY

RESULT	MEAN	COUNT
RECOVERY	11%	15
FREE SPACE	53%	70
OVERLAY CAND	13%	18
GIVE UP	23%	31
HARD REQUEST	0%	0

LAUNCH ACTIVITY AND ADDITIONAL MEMORY MANAGEMENT ACTIVITY SUMMARY

	PROCESS LAUNCHES	PROCESS SWAP-INS	PROCESS PREEMPTS	MEMORY ALLOCS	SPECIAL REQUESTS	MM I/O READS	MM I/O WRITES	RELEASE DATA SEG	RELEASE CODE SEG	CLOCK CYCLES
COUNT	4664	117	650	132	9	87	63	50	73	7
RATE	72.2	1.8	10.1	2.0	.1	1.3	1.0	.8	1.1	.1
MAX RATE	79	6	12	6	0	4	3	3	4	0

SUMMARY OF DISC ACTIVITY

	COUNT/RATE				MAXIMUM RATE (USER/MM)		
	ALL I/O	READS	WRITES	CONTROL OPS	READS	WRITES	CONTROL OP
ALL DISC	4024/ 62.3	1389/ 21.5	2611/ 40.4	24/ .4	22/ 4	44/37	1
DISC 1 (LDEV 1)	972/ 15.0	343/ 5.3	622/ 9.6	7/ .1	6/ 3	12/17	0
DISC 2 (LDEV 2)	889/ 13.8	305/ 4.7	581/ 9.0	3/ .0	6/ 0	10/14	0
DISC 3 (LDEV 3)	2163/ 33.5	741/ 11.5	1408/ 21.8	14/ .2	12/ 0	24/ 9	0

SUMMARY OF LP ACTIVITY

	COUNT/RATE		MAXIMUM RATE	
	ALL I/O	WRITES	CONTROL OPS	WRITES	CONTROL OP
ALL LP	0/ .0	0/ .0	0/ .0	0	0

SUMMARY OF TAPE ACTIVITY

	COUNT/RATE				MAXIMUM RATE		
	ALL I/O	READS	WRITES	CONTROL OPS	READS	WRITES	CONTROL OP
ALL TAPE	0/ .0	0/ .0	0/ .0	0/ .0	0	0	0

TEST 4

INTERVAL LENGTH: 134.355 SECONDS (2.2 MINUTES)

CPU ACTIVITY SUMMARY

CPU STATE	MEAN	MAX	LENGTH	COUNT	TOTAL TIME
CPU BUSY	23%	42%	.007	4267	31.463
PAUSE DISC & SWAP	7%	13%	.032	281	8.956
PAUSE DISC	36%	53%	.023	2111	48.662
PAUSE SWAP	3%	5%	.040	102	4.082
PAUSE IDLE	10%	14%		0	12.830
GARBAGE COLLECTION	8%	13%	.016	678	10.907
MEMORY ALLOCATION	4%	9%	.008	667	5.626
ICS OVERHEAD	9%				11.829

MEMORY ALLOCATION SUMMARY

RESULT	MEAN	COUNT
RECOVERY	14%	111
FREE SPACE	62%	484
OVERLAY CAND	17%	131
GIVE UP	7%	51
HARD REQUEST	0%	3

LAUNCH ACTIVITY AND ADDITIONAL MEMORY MANAGEMENT ACTIVITY SUMMARY

	PROCESS LAUNCHES	PROCESS SWAP-INS	PROCESS PREEMPTS	MEMORY ALLOCS	SPECIAL REQUESTS	MM I/O READS	MM I/O WRITES	RELEASE DATA SEG	RELEASE CODE SEG	CLOCK CYCLES
COUNT	4267	667	217	842	114	614	427	252	248	34
RATE	31.8	5.0	1.6	6.3	.8	4.6	3.2	1.9	1.8	.3
MAX RATE	37	10	5	13	2	9	4	4	5	0

SUMMARY OF DISC ACTIVITY

COUNT/RATE

	ALL I/O	READS	WRITES	CONTROL OPS
ALL DISC	4126/ 30.7	2885/ 21.5	953/ 7.1	288/ 2.1
DISC 1 (LDEV 1)	2037/ 15.2	1729/ 12.9	251/ 1.9	57/ .4
DISC 2 (LDEV 2)	919/ 6.8	564/ 4.2	286/ 2.1	69/ .5
DISC 3 (LDEV 3)	1170/ 8.7	592/ 4.4	416/ 3.1	162/ 1.2

MAXIMUM RATE (USER/MM)

READS	WRITES	CONTROL OP
22/ 9	10/61	3
15/ 3	2/35	1
9/ 2	3/21	1
3/ 3	6/17	2

SUMMARY OF LP ACTIVITY

COUNT/RATE

	ALL I/O	WRITES	CONTROL OPS
ALL LP	0/ .0	0/ .0	0/ .0

MAXIMUM RATE

WRITES	CONTROL OP
0	0

SUMMARY OF TAPE ACTIVITY

COUNT/RATE

	ALL I/O	READS	WRITES	CONTROL OPS
ALL TAPE	0/ .0	0/ .0	0/ .0	0/ .0

MAXIMUM RATE

READS	WRITES	CONTROL OP
0	0	0

Ok here