BoB Wolf / FONAR

SECRETS OF SYSTEM PERFORMANCE UNDER AOS

By Ron A. Fitch

Note: Unless otherwise noted, any references in this manual to AOSWATCHER, or "the Watchers", also apply to the DESKWATCHER Performance Monitor as well.

6-SEP-89

1. dont use ACL in SYSGEN,
   set ACL on # TO +, OWARE

2. 12-char filenames

3. delete un-used files in @ during UP macro

4. Re-order PSFiles macro - all load file one
   after another DUMP/LOAD

5. AOS ONLY
   a) Is gHOST.OL contiguous?
   b) patch OTMIN in SYS.SY? (Pg 6-6)
   c) re-locate SWAP.SWAP (Pg 1-26)

6. locate BIT MAP in middle of disk ~20 MB

7. Watch hash frame size - O+ASY+ may all hash
   to same hash block - write program PG 4-7.

Table of Contents

Preface

This manual has come about after several years of "suffering" with mediocre AOS performance and watching others suffer. As a result of teaching many AOS Internal Structures and AOS System Management courses, it became very clear to me that too many AOS installations don't achieve nearly the optimum performance available from their AOS system. This is a tragedy because in my mind AOS is probably one of the best all-around operating systems extant in the mini world today.

So, in response to the numerous requests to provide "inside information" concerning AOS performance and its improvement, this manual has been produced to keep you people quiet. Now, while this material is meant to be used in conjunction with the **Deskwatcher** and **Aoswatcher** utilities, I trust that for those of you without this software your needs will nevertheless be satisfied. Utilities aside, I simply cannot supply all the answers. I would however like to provoke you into doing a little research on your own as the need arises; and, I would love hearing about your results. I will be happy to consider passing your findings along in future revisions of this Performance manual.

On a different note, regarding this manual exposing "inside information", what you will discover is that most of this so-called "inside information" is actually very public and very obvious. Despite the obvious nature of this data, don't kick yourself for not having figured it out before; sometimes things are overlooked because they are so obvious.

A side note should be made here which is that this material is intended for a Sysgen'able AOS system and not necessarily a Desktop Pregen System. While most of the material will probably apply to a Pre-gen system, no guarantees are made in this respect.

This manual was originally conceived in the Spring of 1985 and was published as: "Secrets of Desktop Performance under AOS. It has been updated under this new title to accommodate new findings. Special consideration in Chapter 6 has been given to deal with issues concerning the Desktop, obviating the need for separate manuals.

Many thanks are due for the creation of this manuscript; too numerous to mention here. Special recognition however should be given to William Wilson, founder of Innovative Data Systems and Innovative Press, and Walt Howard of Sam-Walt Productions. It is through their keen insight, support and generosity that I have been able to develop the original **Deskwatcher** and **Aoswatcher** System Performance Monitors while at the same time conducting the research leading to the generation of this manual.

So, relax, unblock your mind and be ready for a few "aha's" to pop up along the way. If you have any interest in improving the performance of your desktop AOS system, the time spent reading this material will be well worth your investment.

Good luck and Happy Tweaking!

Playfully Submitted,

Ron

Ron Fitch
Redondo Beach Calif. - July 1986

Introduction
Chapter 0

What is System Performance?

Dealing with System Performance is one of the most important aspects of Computer System Management; especially when dealing with Desktop Generation Systems. Luckily, controlling System Performance follows the same "laws", whether you are on a Desktop System or an M/600.

Now in order to spend time looking at System Performance issues, we first need to understand what System Performance really is. To assist us in this definition, I have turned to a few of the acknowledged "masters" on the subject; namely: The famous Harry Katzan, Jr.; William S. Davis, from Miami University; and Liba Svobodova of M.I.T. fame. References to their works are in the bibliography.

Svobodova says, "....loosely defined, System Performance is the degree to which a Computer System meets the expectations of the person[s] involved". Of course, people may demand too much from a system; like, compute me the national debt in 5 minutes... He then offers a more precise definition: "performance is the effectiveness with which the resources of the Computer System are utilized toward meeting the objectives of the software system".

According to Harry Katzan, System Performance is: "...the measure of how efficiently a Computer System operates and is a function of: thruput, response time and availability. Davis adds "reliability" to the above list of functions. I will exploit these four functions in a moment. Before I do however, I need to clear a distinction with you: the difference between System Performance and System appropriateness.

Performance isn't the only factor in choosing a "system", system appropriateness is just as important; if not more so. Unfortunately, thanks to clever salesmen and SE's, System Performance has become emasculated, overriding other more [initially] prominent issues. A system may posses excellent performance characteristics, yet if it is inappropriate to the task(s) set before it, all the performance improvement in the world won't make any difference.

Performance can be broken down into two distinct viewpoints:
- Individual Applications
- Overall System Effectiveness

Usually when we discuss System Performance we examine the second viewpoint; whereas, users of application programs usually concern themselves with the first viewpoint. In reality, what we need is to effect a compromise between user applications and overall system performance. Although that is the goal of this manual, we will naturally be focusing more on the issue of overall
system performance because it is less understood. In certain respects, improving individual application performance requires nothing more than common sense or intimate application knowledge.

Be forewarned, there may be a nasty catch to performance improvement. Axiom 2 of "Fitchmouse's Laws of Computer Dynamics" states: "Improving System Performance in one area is often done at the expense of performance in another area - like the pollution caused by pesticides". Don't necessarily take this too seriously. It is meant only as food for thought.

The Functions of System Performance

Now then, let's define the functions of System Performance. Remember, the components of System Performance are: Thruput, Response Time, Availability and Reliability.

Thruput

We can define Thruput as the total volume of work performed by a system in a given period of time. It is a good measure of the efficiency of the Computer System.

Response Time

The most useful definition for response time that I can think of is: The time between when a user makes a request for data and when the system actually responds.

Availability

Availability is probably best defined as a system's ability to service requests for information with no delay other than that generated by the response time. Availability is a measure of a Computer's effectiveness to the user(s).

Reliability

Reliability is the guarantee that the Computer System will be available when needed to process data. In the long run, reliability is the most important measure of system effectiveness, whereas the other three aspects of System Performance are more important in the short-term. When cash is tight, we tend to back off on the frequency and quality of Hardware/Software support and maintenance. This is actually gambling in favor of immediate, short-term survival rather than consider the "big picture".

Divisions of System Performance

Under AOS, we can actually represent the four functions in a different way. This manual covers the 6 main divisions of System Performance in an AOS environment. Briefly speaking they are:

- Hardware Configuration and Add-ons
- Memory Management and Process Swapping
- Process Scheduling and Management
- I/O and [more importantly] the Disk World
- Altering Process Privileges
- The GHOST Context in conjunction with PMGR

Note: The order of the above 6 items is not necessarily meant to be suggestive of their relative importance. For example, that I discuss Hardware configuration last in this manual is not to imply that it is of trivial concern. In fact, it may be the most important aspect of System Performance.

Performance Overlap

As a additional point, don't be surprised to see an overlap in performance improvement methods. At times, improving performance via one technique may impact the system in other ways; as we saw in Axiom 2 of Fitchmouse's Laws of Computer Dynamics. Don't be afraid to experiment. After all, experimentation is what System Performance is all about.

In general, things we can do to improve System Performance include:

- System Configuration (AOSGEN)
- Operation Management policies in the user environment
- Proper selection of Hardware and Add-on devices
- Altering the System Load
- Rewriting inefficient software or (if the sources are not available) altering its runtime implementation by adjusting such parameters as:
    - Process Priority
    - Process Residency
    - System Process Load

As a side note (that seems to fit nowhere else in this manual so I'll originate it here), it is important to understand the order in which processing is handled on an AOS system. Generally speaking, the Processing Priority Order is: Hardware, System Software, Application Software, Benefitted User (human). Specifically, the order is:

- Hardware Instruction Cycles
- Hardware Interrupts
- Interrupt Service Routines
- System Call processing/completion
- Process/Task Scheduling
- Application code directly benefitting the user

So, with some of the basic definitions out of the way, let's get on to the specifics. Before diving in however, I recommend that you spend some time "meditating" on the points discussed thus far. I can guarantee you that there is more here in these four pages than [initially] meets the eye.

Having done that, then hold your breath, and away we go.....

```
***********************************************
*                                             *
*                                             *
*               Chapter   1                   *
*          AOS  Memory Management             *
*                                             *
*                                             *
***********************************************
```

Memory Management under AOS

Introduction

Memory is the most important system resources, with the Disk world running a close second. Yet while being the most important resource, it is the one most poorly utilized. If you want to really make a difference in System Performance, you need to have a thorough understanding of memory management theory under AOS and then put it to use! All the theories in the world are worthless if they are not exploited to their fullest potential.

The Basics

In order to discuss Memory Management, we need to first clear up some basic concepts concerning the subject. Let's start with a few simple ones.

Although Eclipse systems are essentially word-based machines (a word being 16 bits, 2 bytes), AOS is a page-oriented Operating System (a page being 1,024 words - 2000 Octal). The Kernel of AOS occupies the lowest physical memory of the machine, while the Dynamic portions of AOS (GVMEM) and user processes occupy the remaining memory.

Logical .vs. Physical Address Space

Under AOS, each process is allowed a logical address space up to 32kw; AOS has similar restrictions. A process's Logical Address will always be the same (unless changed by the program code), while the corresponding Physical Addresses are subject to change at anytime; especially if the process ends up being swapped to disk.

Because the hardware supports only a 15-bit program counter (PC), the highest addressable memory location is Logical 77777 octal. In order to address all the physical memory (which requires an address length of 20 bits), the Logical Addresses must be **translated** into Physical Addresses.

Address translation is accomplished with ease by the MAP (Memory Allocation and Protection) or MMPU (Memory Map and Protection Unit); an integral part of every Eclipse machine intended for AOS use. Generally speaking, this translation unit (herein referred to as "The Map") consists of 2 or 4 user maps - A & B, or A,B,C,D - and four Data Channel maps - A,B,C,D - of which only A & B are utilized by AOS. For the Model 10 Desktop, there is also an 8086 Map to address up to 512k of memory.

Diagram 1-1 below outlines the Logical to Physical address translation process which is detailed in the section:

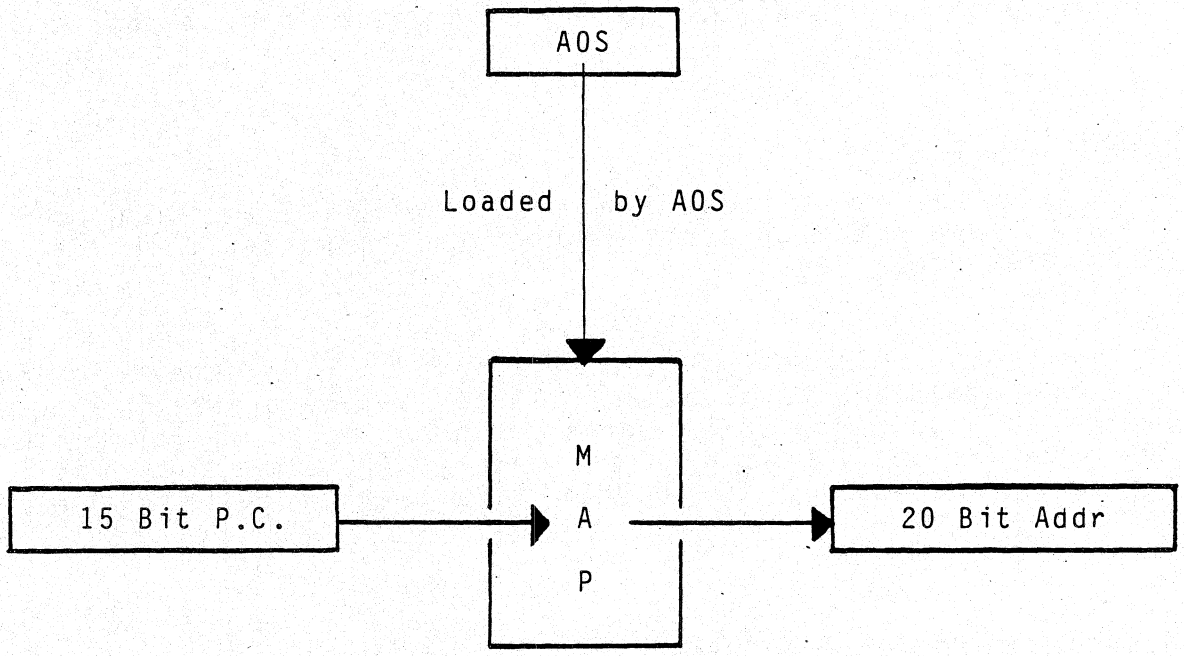


Diagram 1-1
Address Translation Overview

MAP Unit Components

Diagram 1-2 illustrates the MAP Unit components, however this can be misleading. The MAP Unit is not exactly a "unit" per se', it is a collection of micro-code routines that come together functionally as a "unit". The MAP unit is an integral part of the system hardware design; unlike other systems.

The highlights of the MAP Unit are:

- Performs Address Translation
- Implements the various Hardware Protection Features
- Integrates the 8086 with the system (DG/10 Desktop only)

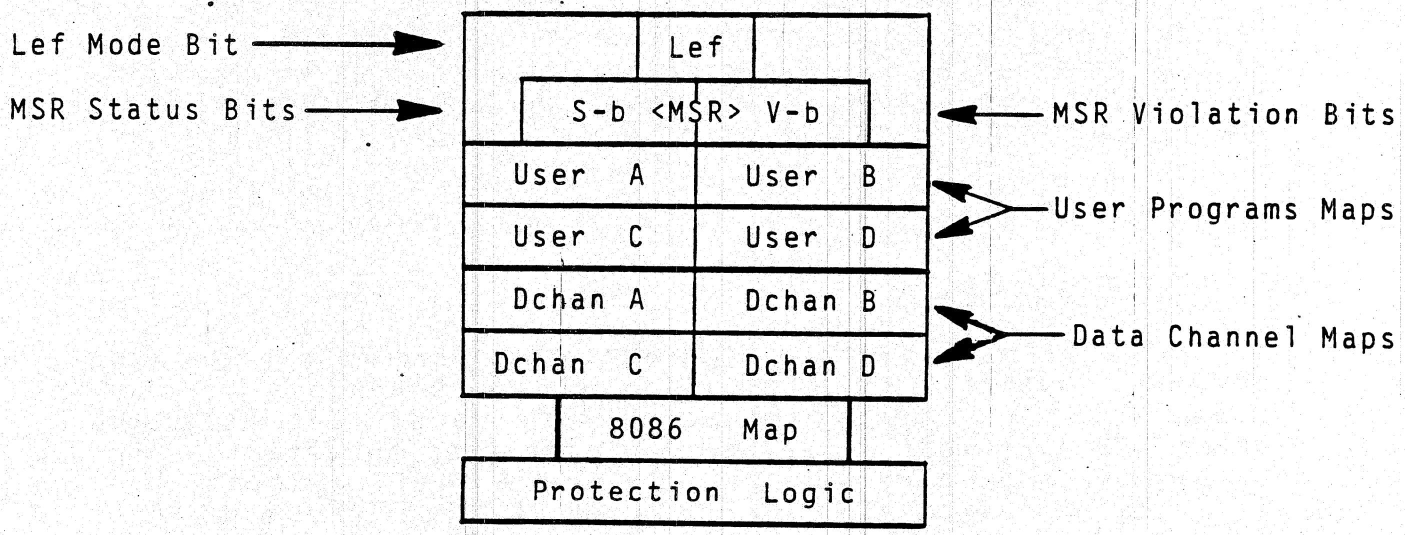


Diagram 1-2
Block Diagram of the MAP Unit

Map Status Register - MSR

The Map Status Register (MSR) is a 16 bit register used to set and interrogate the MAP status. Setting the MAP status is used to [programmatically] enable a user map, a data channel map, or on the Model 10 Desktop, the 8086 map. The status register allows the protection features to be enabled. Reading the [so-called] S-bits (of the MSR) is done by the Interrupt World in order to remember "who" (ie. which user map) was in control of the CPU (ie. enabled) when the interrupt sequence occurred. This way we are guaranteed that we can reschedule to the correct user process with the least amount of difficulty.

The Map Violation bits are interrogated in order to determine the reason for a MAP protection fault (often called a Map Trap, a Protection Fault, or just a Trap). When a Protection Fault occurs, the hardware disables the map, updates the MSR and finally jumps to the AOS Map Protection Fault Handler. This Fault Handler reads back the Map Status Register to determine "who" faulted and why. If AOS (under Map B) should trap, a System Panic will usually occur.

The "official" AOS response to a user trap is to terminate the "offending" process with an IPC message sent back to the father process indicating the cause of the trap and the contents of all relevant hardware registers. AOS creates a .Brk file in the process's current directory as well. For program detected errors, a breakfile can be created with ?BRKFL system call.

The LEF Mode Bit

This bit (in the MSR) indicates that LEF instruction mode is enabled. With LEF mode enabled, all machine I/O instruction codes will be interpreted as [single-word] LEF instructions. Under AOS, LEF mode is enabled for the user by default. It can be manipulated directly via the ?LEFE/?LEFD system calls and interrogated with the ?LEFS System Call. These calls are described in the AOS Programmers Reference manual.

The User and Data Channel Maps

Each User or Data Channel map contains 32 registers, better known as "slots". One slot corresponds to 1Kw of Logical Address space. A slot contains the physical page address for the memory corresponding to the logical page address. AOS loads the slots for User **Map A** every time it dispatches to a User Process. When referencing its own Virtual Memory (GVMEM), AOS loads and enables User **Map B**. User Maps C & D (if they exist in the CPU) are not used by AOS. Only one user map may be enabled at a time.

If all user maps are disabled, the machine is said to be running in unmapped mode. Although unmapped, AOS can always map its 31st page to any physical page in memory as needed (using the Map Spvr Block 31 feature of the hardware) in order to access user code and control blocks.

Data Channel maps are enabled by the AOS device driver logic before I/O to a DMA-type device is actually started. This allows the DMA device (disk or tape) to perform I/O to/from any page in memory concurrent with program execution. A Data Channel map can be enabled concurrent with a User Map.

The 8086 Map

The 8086 Map is used in conjunction with the 8086 processor on the Model 10 Desktop only. This MAP is not used by AOS. It contains 512 "slots"; one for each page of memory currently available to the 8086 processor. This map is invoked if the user run the MS-DOS or CP/M-86 Operating Systems on the Model 10.

The Four types of Protection Faults

A Protection Fault will take one of four forms:

- Validity Protect
- Device I/O Protect
- Indirect Address [loop] Protect
- Memory page Write Protect

Validity Protect

A Validity Trap occurs when program code attempts to reference a Logical Page for which there is no corresponding Physical Page. This can occur in programs that have not declared their full 32kw address space as being in use, and then attempt to reference one of those undeclared Logical Pages. Validity Protect is enabled by virtue of enabling a User or Data Channel Map.

I/O Device Protect

An I/O Trap occurs when program code attempts a hardware I/O instruction with I/O Device Protection enabled. Under AOS, I/O Protection is enabled by default for every user process. To disable I/O Device Protection, the user program must issue a ?IDEF or ?DEBL System Call. AOS will respond by clearing the Device Protection Bit in the Map Status Register. Use of the ?IDEF & ?DEBL calls require the Access Devices profile privilege.

Indirect Address (Defer) Protect

An Indirect [address] Protection Fault occurs when the hardware encounters the 16th indirect [address] reference in a single instruction cycle. The assumption is made by the hardware that we are [for all intensive purposes] in a hardware indirect address loop. Without this protection feature, the hardware would loop endlessly attempting to resolve the effective address.

This would not be a problem except that the interrupt bus is not interrogated until the end of an instruction cycle; a cycle which in this case would never end. AOS defaults this protection feature to on and offers no system facility to disable it. In the unlikely event that it should ever need to be disabled, that will need to be done by the user program itself. It will first need device access in order to issue the Load Map Status I/O instruction (which allows disabling of the "defer" protect mode.)

Memory Write Protection

A Write Protection Fault occurs when the user program attempts to modify write protected memory. AOS write protects all shared-code pages to prevent runaway code from accidentally modifying itself. Unshared pages of memory are never write protected by AOS.

Address Translation Theory

As overviewed earlier, Address Translation is the process of "translating" 15 bit "Logical" Addresses into 20 bit "Physical" Addresses. The easiest way to understand the address translation process is to break the Program Counter (**PC**) into two bit-groups: **5 Bits & 10 Bits.**

5 Bits	**10 Bits**

In 5 bits we can reference a number 0-31 (32 Pages per process).

In 10 bits we can reference a number 0-1023 (1024 words per page)

By the time a User Program is in execution, the **A Map** will have been loaded and enabled. As memory is reference by the program, address translation will occur on each reference.

What the MAP does is to break each Logical Memory Address down into its two components: 5 Bits (the map slot #) and 10 Bits (the physical page offset). It then replaces the high-order 5 Bits with the 10 bit content of the addressed map slot giving us a full 20 bit address. In 20 bits we can address any word of memory on the machine. This is illustrated in Diagram 1-3.

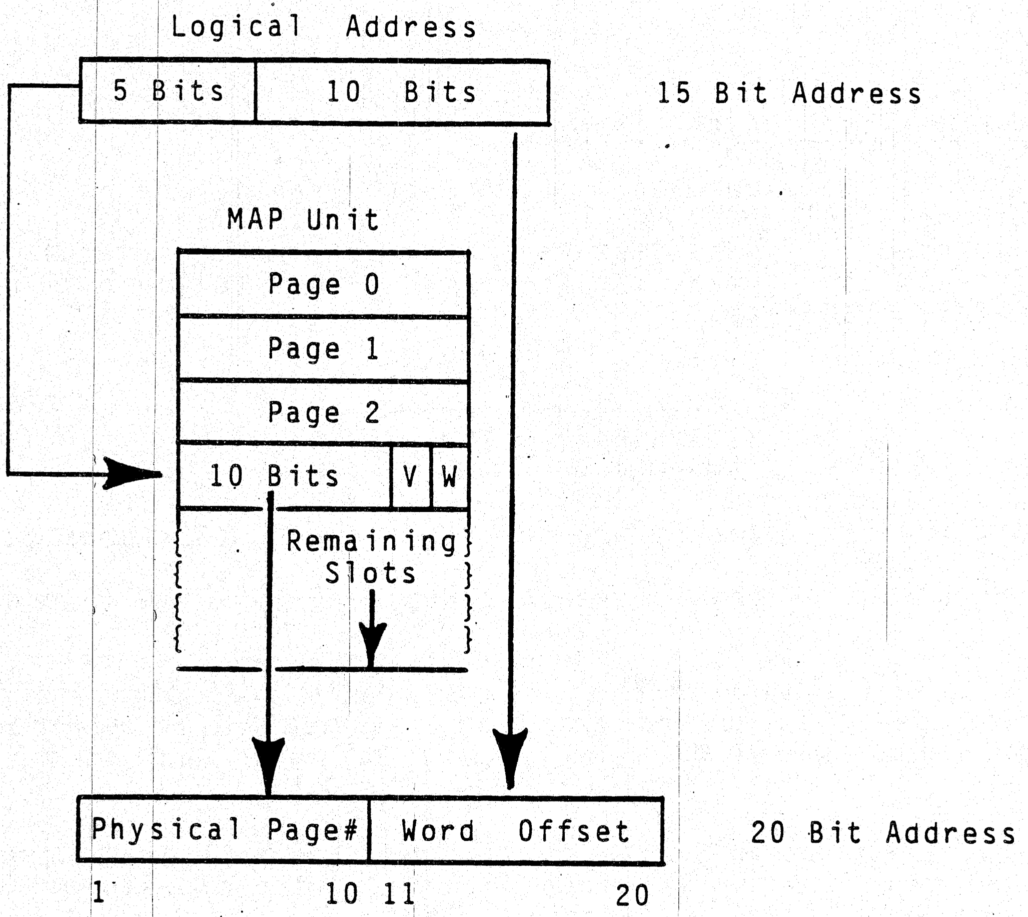


Diagram 1-3
Logical to Physical Address Translation

Page-Sharing under AOS

Thanks to the design of the MAP (diagramed on the previous page), the sharing of pages is a simple thing to do. Pages can be shared within a process or across process boundaries; allowing read and write access (based on what the page will be used for).

Page-sharing can be implicit (defined by MASM or Compiler directive statements) or it can be explicitly requested by the program logic with the ?SOPEN/?SPAGE/?RPAGE System Calls in AOS. Implicit shared pages allow shared-code and are always write-protected by AOS. This is to prevent assembler language code and/or "runaway" code from modifying itself; which would defeat the re-entrant and recursive properties that are available from writing in shared-code. More information on page sharing appears at the end of this chapter.

Page sharing occurs when two or more map slots point to the same physical page. It is quite permissible for two processes to address the same physical page; even pointing to it via different logical page addresses. An example of page sharing is detailed in Diagram 1-4 on the next page. It is this capability that is the power behind the shared page mechanism of the MAP.

Once shared pages are declared to AOS, the Process Scheduler takes over ensuring that the "A" map is loaded just prior to continuing process execution. Should a process change its address space by direct I/O instructions to the MAP, the Process Scheduler will have no way of knowing that this has occurred. When it comes time to restart the process after an I/O interrupt or reschedule, it will reload the "A" Map from the Process Table (PTABLE), reverting the logical address space to its previous state before the MAP I/O instructions were executed. Of course, the MAP state can be altered with the interrupts disabled, but this brings in a host of new problems.

Page sharing is a powerful facility that can assist us greatly in the management of memory. However, if it is improperly used, it can actually waste more space than it saves. Poor memory management will also negatively impact overall system performance. This is caused by AOS being forced to spend a significant amounts of time keeping the memory resource in order. With poor management in effect, this problem could become self-defeating; additional memory contention being created while trying to resolve earlier memory management problems.

Page sharing is a powerful facility. Use it wisely.

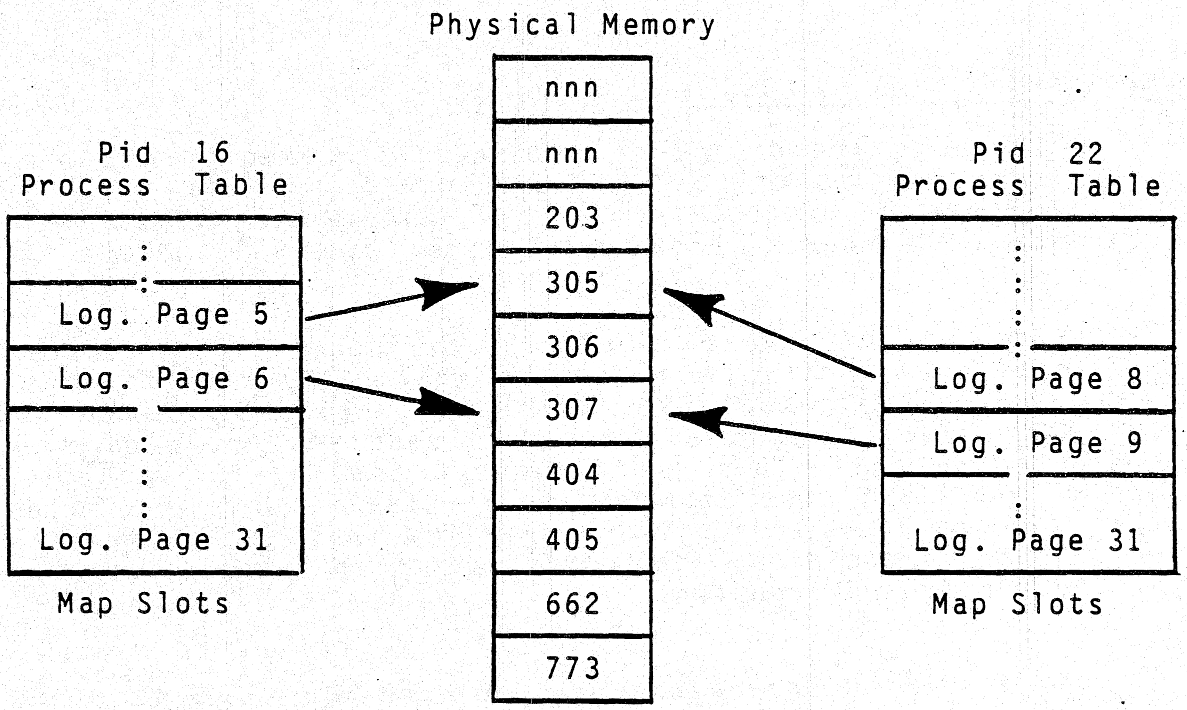


Diagram 1-4
An Example of Page Sharing

Hardware Memory Management Summary

As you can see, Memory Management is partially a hardware function controlled by the AOS Operating System. This marriage of Hardware functionality and Operating System control makes for the greatest thruput for this kind of system design. And, in addition to this functionality, AOS has a set of Memory Management algorithms providing even greater control over the memory resource.

Additionally for you slick Assembler language programmers, you can make use of User Maps C & D (if they exist on you hardware) by reading the CPU technical reference manuals and experimenting with the MAP unit. Be prepared to accidentally "crash" the system the first few times out until you get the kinks out of your logic. Luckily, the crash will probably not be serious. A simple ESD or fixup run and you will be back in business. This kind of experimentation is encouraged. Who knows what you will dream up as a result; maybe an improved System Performance monitor.....

As was pointed out earlier, memory is the most precious system resource. In the "real world" the demands on memory are often more than the system has pages to satisfy those demands. On less sophisticated systems such as the DG/1 Laptop, when the program requirements exceed the available machine memory the program takes an error and terminates. MS-DOS, due to its somewhat primitive design, is incapable of resolving the problem. Fortunately, under AOS there are a number of ways to resolve the problem of inadequate memory. In order to take advantage of these algorithms however, we need to understand what Memory Contention is, as well as what causes it.

What is Memory Contention?

Memory Contention under AOS is a situation where there are more requests for memory than the Operating System can satisfy by allocating pages from the Free Memory Chain (**FMCHN**).

If the memory request is made by a user program or the Ghost, that request will always be in multiples of pages. If the request is made by AOS internally, it could be for an amount of memory as small as 8 words or as large as 1024. Memory segments that are less than a page in length are known to AOS as **Chunks.**

The Four Conditions of Memory Contention

In an AOS environment there are 4 conditions of Memory Contention: Light, Medium, Heavy and Critical! (The 4th term is my own). These levels along with AOS's resolution for them are detailed in Diagram 1-5 below. The details on memory resolution appear later in this chapter. Unfamiliar terms can be found in the glossary and will be defined/detailed later.

Memory Condition	Condx.	AOS's Response to the Memory Contention
Condition 0	None	Aos obtains a free page from **FMCHN**
Condition 1	Light	Aos obtains memory from **OVMCH** and **CANCH**
Condition 2	Medium	Process are swapped off of the **BLKQ**
Condition 3	Heavy	Processes are swapped off the **RELQ**
Condition 4	Critical	Swap-aborts start occurring

Diagram 1-5
Conditions of Memory Contention

How to Detect Memory Contention

As AOS enters Memory Contention, it becomes increasingly evident to the users on the system. By the time they get around to complaining about it however, the system may be deep into heavy contention, or even at the critical stage. At this level, the only way out may be to do a system shutdown and start over. If no steps are then taken to resolve the problem, you are most assured that the problem will return.

The way to deal with Memory Contention is to spot and handle it before it becomes a big issue. Like heart disease, you need to know the warning signs. The signs pointing to Memory Contention are:

- Overall machine activity slows down or develops a jerky "feel ".
- Program load time increases significantly.
- The **MEMIN** Screens of **Aoswatcher** indicate that memory "thrashing" is occurring. (Thrashing is a state where memory is actively being shuffled between the user processes and AOS).
- PED and **Aoswatcher** screens indicate that processes have been swapped to disk.
- The **DSTATS** screen of **Aoswatcher** indicates that the Avg. Seek distance has suddenly and dramatically increased.
- The **DSTATS** screen of **Aoswatcher** indicates that the number of disk read requests have significantly increased.
- The **DSTATS** screen of **Aoswatcher** indicates an increase in the number of disk write requests; probably from writing swapped processes to disk.
- The **MEMIN** screen of Deskwatcher shows that swap-aborts are occurring.
- An AOS System Deadlock.

Each of the above points will be addressed near the end of this chapter when we take a look at what can be done to improve the Memory Management scene under AOS. Stay Tuned.....

The Causes of Memory Contention

Thus far, we have looked at the warning signs of Memory Contention. Now, let's examine the different elements contributing to Memory Contention. I will just list them here. Later we can spend some time looking at their implications separately, and if necessary, in conjunction with other related items.

The apparent causes of Memory Contention are:

(1) Not enough Physical Memory.

(2) Too many processes on the system.

(3) Too many Resident and/or Preemptible Processes.

(4) Processes over-extending their address space via **?SSHPT** and/or **?MEMI** System calls.

(5) Programs doing too many/frequent **?SPAGE** calls and not balancing them with **?RPAGE** calls.

(6) Inadequate use of shared-code in user programs.

(7) Unnecessary devices specified during **AOSGEN** and/or their buffers have been declared too large.

(8) Sync buffers gen'ed and those lines are not used.

(9) Too many Cache Buffers in the system, and/or poor utilization of those buffers.

(10) Buffer size is too large during Dump/Load

(11) Program buffering scheme(s) allocating too many buffers, or buffers that are too large, or both.

(12) Too many batch streams active.

(13) Too many [short-term] demands being made of EXEC forcing its address space to grow. (As of Release 7.0, OP is no longer notified of this via console messages.)

To understand how and why these aspects can impact System Performance, we need to look more closely at the more visible Memory Management structures designed into AOS. This is by no means an exhaustive study, but it offers enough information to assist you later in understanding how to remedy your system's Memory Contention problems.

Let's begin by examining how AOS Obtains a page of memory.

Finding Memory Pages - General

When AOS needs to obtain a full page of memory, it looks to the Memory Management Chains in the following order:

- The Free Memory Page Chain (**FMCHN**)
- The Overlay Memory Page Chain (**OVMCH**)
- The Shared-Page Candidate Chain (**CANCH**)

If AOS cannot obtain a page from one of these chains, then processes will have to be swapped to disk in order to free up the required memory.

The Free Memory Chain

The Free Memory Chain (**FMCHN**) is where pages of memory are relinquished to when no longer needed by the system. Pages are taken from **FMCHN** during Memory Contention Condition 0. Events that will contribute pages to the **FMCHN** are:

- Process Terminations
- A process reducing the size of its address space via the **?MEMI** and **?SSHPT** system calls.
- AOS recombining internal memory "chunks" to make up a full page of memory.

OVMCH and CANCH Chains

AOS maintains two groups of page chains for disk overlays no longer in use (**OVMCH**) and Shared Pages with a zero use count (**CANCH**). Both sets of chains are maintained in LRU order: pages that haven't been used for the longest period of time are made available to the page request routines first; ie. in FIFO order.

As of AOS release 5.0, these chains (as well as some of the process queues) were expanded into regions: 0,1,2. The idea is to determine the "age" of a page and place it on the appropriate region chain. Pages attached to Region 0 have been around the longest amount of time and are therefore available first. Pages living on Region 2 are relatively brand new and are used as a last resort.

In theory, when AOS needs a page and the **FMCHN** is empty, the next check is made to: **OVMCH**-Region 0, **CANCH**-Region 0, **OVMCH**-Region 1, **CANCH**-Region 1, followed by an examination of Region 2 for these chains. I say in theory because all indications (from reading the Memory Management program code) are that Region 2 is not actually used. Any answers Data General?

Finding Memory Pages - Detail

When AOS needs a free page of memory and none are available on the **FMCHN**, we then enter into Memory Contention. System Performance suffers in direct proportion to the level of Memory Contention. For purposes of discussion, I consider pulling a free page from **FMCHN** to be Memory Contention Level 0.

To reiterate, AOS to obtain a page of memory (for whatever reason) looks in the following areas:

Contention Level 0	-	FMCHN
Contention Level 1	-	OVMCH (Region 0)
	-	CANCH (Region 0)
Contention Level 2	-	Swap process(es) from MBLKQ (Region 0)
	-	Swap process(es) from BLKQ (Region 0)
Contention Level 3	-	OVMCH (Region 1)
	-	CANCH (Region 1)
Contention Level 4	-	Swap process(es) from MBLKQ (Region 1)
	-	Swap process(es) from BLKQ (Region 1)
Contention Level 5	-	Swap process(es) from RELQ

Prior to Release 5.0, we emptied the OVMCH and CANCH chains before resorting to swapping processes out. As of release 5.0 this philosophy has been changed somewhat, as can be seen from the above list of Memory Contention Levels.

Speculation

As mentioned earlier Region 2 although present in the system is not being utilized at this time. Logic would dictate that if it is ever incorporated into AOS Memory Management, that it will be accessed before Condition Level 5. We only want to swap processes off of RELQ as a last resort.

The Purpose of Preemption

As we enter deeper into Memory Contention, processes begin to swap. Another term for process swapping is known as preemption. As pointed out earlier, when Memory Contention is so bad that we have to start preempting processes, System Performance begins to suffer drastically.

Although preemption does impact System Performance, its goal is actually to improve performance in the long run. This purpose will hold true until enough demands are made on the memory resource to force the machine into Heavy and Critical Memory Contention. At these levels, preemption serves to keep the machine away from "system-deadlock".

Preemption is accomplished by removing blocked processes from memory one at a time until the desired memory segment is pieced together from the pages released to **FMCHN** as the process-(es) are swapped to disk. Preemption begins in the condition of Medium Memory Contention, attempting to return the machine to a state of Light Contention. Selected for preemption are the oldest processes on the **MBLKQ** and the **BLKQ** (starting with Region 0)

Preemption vs. Swap-Abort

The only condition worse than preemption is know as a swap[-in] abort. A swap-abort is where AOS begins to swap a process back into memory (requiring substantial system overhead) only to find out that another process code-path within AOS has stolen some or all of the memory just freed, or the process targeted for swap-in has been blocked again. Swap-aborts essentially mean the system is doing a lot of extra work for nothing.

Preemption Rules

In order to preempt a process, we need to examine a couple of parameters with regard to the process "requesting" the preemption vs. the process to be preempted (otherwise known as the target process). The main parameters to consider are a process's type and its PNQF (see Chapter 2). For example, a requesting process can never force a superior process type to swap (Ex: a swappable process attempting to preempt a preemptible process). To understand the preemption rules, refer to Diagram 1-6.

		TARGET PROCESS		
		R	R	S
Requestor Process	R	Never	Always	Always
	P	Never	TB or TLP	Always
	S	Never	TB	TLP

R= Resident   P= Preemptible   S= Swappable

TB= Target Process is Blocked

TLP= Target Process has lower PNQF

Diagram 1-6
Preemption Rules

Technical Indications of Memory Contention

While researching this chapter of the book, I decided to set up an experiment that would allow me to really observe what happens when AOS is deep in the throes of heavy memory contention. To do this, I removed one of the 256kw memory boards from my Model 10 Desktop; reducing the memory from 640kw down to a mere 384kw. (For a system with as much activity on it as my Desktop, 384kw is a mere fragment of memory.) Next, I initiated about 20 processes; most of them compute-bound memory hogs, as well as a single process (appropriately named Glutton.Pr) specially designed to steal huge amounts of memory away from the AOS Memory Pool; I requested 200kw for the experiment. Monitoring the system with the **Deskwatcher** and PED utilities produced interesting observations.

First off, the number of Swappable processes in core was constantly in a state of flux; ranging anywhere from 15 to 19. That swapping was occurring in the machine was also evident by watching the "flicker" of the SPU Led's and listening to the chatter of the winchester disk. Within about 8 minutes, the number of swap-ins climbed drastically to 1,000+ - I knew then (snicker.) that AOS was having difficulty.

Another visible effect of the Memory Contention was that the average seek distance for the disk rose dramatically. Normally, 50 - 70 cylinders was the average seek on this system. But now, the average was sitting at 149! This means that at times it was probably even as high as 200+ cylinders.

A final experiment increased the memory requirements of Glutton.PR to nearly 300kw. In this experiment, it took an average of 35 seconds to load and begin execution of a program about 14kw (usually PED or Deskwatcher). A few times when this experiment was tried, AOS would end up in a perpetual memory contention; otherwise known as a **"System Deadlock"**.

Perpetual memory contention is where process preemption creates a "rippling" effect and is never really resolved. At such times, AOS spends over 95% of the machine's time rescheduling processes and attempting to resolve the memory contention caused by the rescheduling; ad-nauseam. The only way out of this extreme situation seems to be with an emergency shutdown (**ESD**); unless you have a <u>high</u> priority <u>resident</u> process available to terminate the "offending" process(es).

The above experiments were chosen as they simulate real world situations. Only when properly identified can problems be solved.

Resolving Memory Contention

How to resolve Memory Contention depends on the predominate level of Memory Contention being experienced. There are a number of obvious solutions however that are useful to know. Let's take a look at them.

Adding Memory

In a large percentage of cases, Memory Contention can be easily resolved by adding additional memory. Typically, adding memory will take a machine that is in heavy or critical contention and return it to light or medium contention. Unfortunately, some user software is designed to gobble up as much memory as is available, making this solution only a temporary one.

Although this solution can be a costly one in terms of dollars, in the long run it may actually be cheaper and less of a hassle than spending potentially hundreds of person-hours working out other solutions. If adding memory doesn't seem to help, or, if you are already "max'ed out" memory-wise, then other solutions are necessary.

Process Overload

Memory contention problems on your system may be caused by forcing too many processes to run on the system. Using the ?.CLI macro, PED and/or the **Aoswatcher** process screens may establish that there too many processes on the system or that a number of the processes are making excessive memory demands. Although in theory AOS supports up to 64 processes, in actual practice the number is much less.

In the case of too many processes, take a process inventory and be certain that you can justify **every** process being on the system and of its process type. Remember, although a process may be blocked, its very existence in the process tree (swapped or not), places demands on the AOS memory space. This is because for every process on the system, AOS is internally maintaining a 96. word Process Table (Ptable) along with a 100. word Ptable extender used when the process is resident. Ptable and Extender space is allocated in page multiples. If pid fragmentation exists, AOS may be using only a portion of <u>several</u> allocated Ptable pages.

Reducing process size can only be accomplished by a competent System Programmer. Programs written in High-Level languages typically require approximately 35% more memory space than those written in assembly language. Having a programmer "tighten" sloppy code can dramatically enhance System Performance if that program (as well as others like it) is frequently utilized by several users at a time. More on this later.

Resident/Preemptible Processes

Although marking a process as Resident or Preemptible can initially improve its own performance, too many Resident and/or Preemptible processes can force AOS into a memory deadlock, requiring at minimum an ESD to resolve it.

A process should only be declared Resident if one of the following conditions exist in the program code:

- It contains **U**ser **D**evice **D**river code.
- It utilizes the [undocumented] ?SPY, ?PCREATE, ?PRELEASE or ?PMAP System Calls.
- It addresses memory pages physically.
- It uses the following System Calls: ?AMAP, ?DEBL, ?HIST, ?IDEF, ?IMSG, ?IXHIST, ?IXIT, ?IXMT, ?STMAP
- It is an IPC-driven communications board driver (such as the PMGR)

A process should be declared Preemptible only when you want it to possess a high process priority but don't require resident properties. Preemptible processes are swapped only under conditions of heavy memory contention. Process types are explored in Chapter 2. - Process Scheduling.

Batch Streams

Batch streams allow processes to run unattended and are handy for doing program compiles/assemblies/links while editing other source code. Unfortunately, batch streams can take an additional chunk out of the system memory pool; specifically during program development cycles. If a batch stream invokes a compile, Link, Aosgen or some other utility, memory demands are increased by the batch stream as well as the utility program.

A batch stream merely running CLI commands typically requires very little memory space because the CLI is written predominantly in shared-code. Usually, one batch stream is sufficient, with two being a good upper limit.

Process Address Space Declaration

Early on in this chapter we looked at the ramifications of processes that are declared with too large an address space. That a process is "hogging" memory is not always evident however. At runtime, a process may elect to extend its address space via ?SSHPT, ?MEMI or undocumented Desktop ?Pxxx calls (which are not limited to just the Desktop). PED and the **Aoswatcher** process screens will shed some light on this problem.

Using the **Aoswatcher** Memory Information Screen (**MEMIN**), this address space extension may also show up as an excessive amount of User Memory being utilized. Some processes may temporarily increase their shared memory size (?MEMI) or the Shared Partition size (?SSHPT) - which could suddenly force the system into memory contention - only to release that memory shortly thereafter.

Processes using Shared Page I/O (?SPAGE) are often a source of memory contention. Shared page I/O is meant to be utilized when several processes require shared access to disk file(s). Typically however, programmers often utilize shared page I/O to circumvent the process 32kw address space limitation. I have done this by creating a multiple-page dummy file, using the Shared Pages (containing the empty blocks) to declare large arrays at runtime. When used in this manner, Shared Page efficiency data (as reported by **Aoswatcher**) may be of little value.

Although Shared Page I/O requests may not be excessive, another problem caused by ?SPAGE occurs from a process reading in a large number of shared pages and then not releasing (?RPAGE) those pages when finished with them. This can be observed as a large quantity of Shared Pages in use by the system. Inefficient Shared Page usage can cause excessive Disk I/O in addition to contributing to memory contention.

Using Shared-code

Writing user software in shared-code is an excellent way to reduce memory requirements on the system, relieving the Memory Contention somewhat. Writing in shared-code of course assumes that several users will be running the software simultaneously. If not, the time and expense to write in shared-code becomes a complete waste of time and effort.

Shared-code assures that the shared pages of the program physically exist only once in memory, as long as the same .PR file is executed by <u>all</u> users of the program. If several copies of the .PR file exist, they will <u>not</u> all share the same code. The sharing of code is based completely on the pathname to the .PR file.

A classic example of shared-code is of course the CLI. A process running CLI normally utilizes 18 shared pages and only 3 unshared pages (although Pid 2 OP:CLI seems to be an exception). CLI users typically require only 3 pages from the memory pool.

As a final note, shared-code DOES have a slight liability. The LINKer starts the shared-code partition on a page boundary which could waste nearly a whole page of memory per process if the program is poorly planned. Multiply that by, say, 20 users and you have a significant chunk of memory being wasted. Sometimes all it takes is 20kw to throw the machine into memory contention.

More information on writing with shared-code can be found in the various DG programming reference manuals, as well as the LINK/LFE reference manual. These are listed in the bibliography.

AOSGEN Solutions

Learning the ins and outs of AOSGEN can make a significant difference in your system's performance. To fully understand what is available from AOSGEN, [re-]read chapters 3 & 4 of the "How to Load and Generate...." manual thoroughly. It will be time well spent.

On Desktop systems, many sites choose to run the Pre-Gen version of AOS instead of the Sysgen version; usually because they don't understand Aosgen. I have yet to meet someone who understood AOSGEN and chose a Pre-Gen AOS system anyway. "It is just not logical", said Mr. Spock.

With AOSGEN, the most common mistakes you will make are:

- Including unnecessary devices in the .Sy file or declaring their buffer sizes too large.
- Including Sync Buffers when no sync lines are used.
- Specifying too many cache buffers.

Solution?:

To begin with, **never** gen' devices that you don't need. Nothing is accomplished by doing that; unless you know of some off-beat programming trick that I don't..... When you do include a device, choose the buffer sizes carefully. For example, output devices rarely require an input buffer so this can be set to two bytes (1 word); the minimum requirement. Reverse the scenario for input-only devices.

If your system doesn't use sync lines (most systems don't), you can probably Aosgen this parameter to its default of one line.

If you have "core to burn", gen'ing in all 128 Cache Buffers can significantly speed the system disk I/O. On systems with tighter memory pools, this speed comes at the expense of Memory Contention, which of course can result in added disk I/O to the SWAP.SWAP file. So, while there may be slightly faster I/O, the number of R/W requests may increase due to process swapping.

Dump/Load and Program Buffering

When doing Dumps and Loads, it is often desirable to use the maximum buffer size available for the device. Doing this will not only speed the I/O but will also allow more data to be packed onto media such as magtape; if that's what you're dumping to.

The liability of large buffer sizes during Dump/Load is that it can force memory contention. I did an experiment to prove this out. On my Desktop, the buffered devices include: 2 Floppy drives, A Cartridge tape and a Streaming Magtape; not to mention, the two Winchester disks themselves.

In my experiment, I proc'ed Glutton.Pr at 256kw and then initiated a series of 6 processes all doing Dumps, Loads and Moves with a Buffer size of 8192 (4 pages per buffer). It was interesting to watch the impact of losing only 24 pages. The system dropped immediately into Medium Memory Contention. With Glutton.PR set at 400kw, the system would vacillate between Medium and Heavy Memory Contention.

With user programs, the same buffering problems exist, except that things can be worse because user-written software can define [potentially] an unlimited number of buffers, all at 8192. I recommend allowing a variable number of buffers switch selectable at program load or run time. The program could be designed to specify many large buffers and buffer sizes during the system slack periods, utilizing memory-efficient buffering during peak operation periods.

Swapfile Placement

As we've already seen, process swapping is an undesirable side-effect of Memory Contention that happens from time-to-time. Because swapping involves I/O, it is in our best interests to speed that I/O as much as possible. There is a myth that placing SWAP.SWAP on a head-per-track disk will improve its performance. Unfortunately, those disks have a considerably slower data transfer rate. So instead, the opposite effect actually occurs.

SWAP.SWAP is a contiguous file, meaning that all of the disk blocks for the file reside physically contiguous on the disk. When allocating SWAP.SWAP if the current file size requested is less than or equal to the existing SWAP.SWAP, the existing file is reused or shrunk (freeing up disk blocks). If the requested space is larger than the existing SWAP.SWAP file, contiguous space is found by scanning the Bit Map[1] for the requested number of contiguous blocks. If the required contiguous blocks are found, the system comes up normally; otherwise, a FATAL SYSTEM ERROR occurs. The only time you can "force" a location for SWAP.SWAP is during system installation or a subsequent disk compression. And even then, it's a little bit tricky.

When the system is first booted, the SWAP.SWAP file is installed just prior to asking the "Initial Load?" question. This puts the SWAP.SWAP file near the "front" of the disk. Once the remaining software is loaded onto the disk, this may not be an advantageous location as the seek distance from those files to the SWAP.SWAP may be rather large.

To minimize seek time, you will want to place the swapfile near the files most commonly accessed by the system. To do this, the operating system needs to be "fooled" into placing the swapfile approximately where you want it. Once you understand the way in which AOS allocates Swapfile space, this becomes a relatively easy process.

1) See Chapter 4 for Bit-Map details.

SWAP.SWAP Specifications

SWAP.SWAP is a contiguous file, meaning that all of the disk blocks for the file reside physically contiguous on the disk. When allocating SWAP.SWAP if the current file size requested is less than or equal to the existing SWAP.SWAP, the existing file is reused or shrunk (freeing up disk blocks). If the requested space is larger than the existing SWAP.SWAP, contiguous space is found by scanning the Bit Map[1] for the requested number of contiguous blocks. If the required contiguous blocks are found, the old SWAP.SWAP is deleted and a new Swapfile is created from these blocks. Otherwise a System Error occurs and AOS terminates.

Force-Allocating SWAP.SWAP

Armed with the above data and technical information about your disk, it becomes relatively easy to "force" SWAP.SWAP to a new location.

The first step is to override default specs (during System Installation) allocating a swapfile purposely under-sized. Next, perform the Initial System Load. When you get the CLI prompt, create a dummy file with an element size big enough to "gobble up" all the disk blocks up to the point where you wish the "new" SWAP.SWAP file to start.

Now, shutdown AOS, reboot and override default specs (if necessary), setting the swapfile to its new (probably original default) size. Voila! AOS is forced to create the new Swapfile in the "middle" of the LDU. As a final step, delete the dummy file and load your remaining files. They will install "around" the new SWAP.SWAP.

The above procedure has made quite a difference on many a system. You may not be able to eliminate the swapping caused by heavy Memory Contention, but you **can** reduce its impact. Monitoring the average seek distance on the swapping drive will tell you how good a choice was made for the swapfile location.

Memory Contention Summary

As we have seen, memory contention has many causes and there are many cures. Although we have delved deeply into memory management, there is still much to be learned. Aoswatching the memory management can make all the difference in the world.

---

1) See Chapter 4 for Bit-Map details.

```
*************************************************
*                                               *
*                                               *
*                   Chapter   2                 *
*              AOS Process Scheduling           *
*                                               *
*                                               *
*************************************************
```

Chapter 2
AOS Process Scheduling

Processes and Process Scheduling Theory

Process Scheduling is at once fascinating in addition to being a very complex issue. Before we can adequately delve into this topic however, we need to first review some "basics" about processes under AOS.

Process Basics

AOS is Multi-Process, Multi-task Operating System supporting up to 64 processes. Per the glossary, a Process is simply an address space that contains code and data. A process is not a program; a Process "houses" a program. I know this is nit-picking on the definitions, and, it helps to eliminate a confusion that will probably cause problems later.

Technically speaking, there are actually 65 pids. The 65th pid is Pid 0, used by AOS itself. It should come as no surprise that Pids 1 & 2 are reserved for PMGR and OP:CLI respectively. What is sometimes overlooked is that Pids 1 & 2 can actually be any program you like, as long as the programs are named PMGR.PR and CLI.PR respectively. This of course does require some thought and doing, but it is possible. Before attempting a feat like this however, you had better be well versed in AOS internals or you are sure to make a mess of things.

Processes can be of three types: Resident, Preemptible, or Swappable. Resident Process share a priority structure with Preemptible's and will never be swapped to disk. Preemptible processes are the final candidates for swapping if AOS enters into heavy memory contention. They run at a higher priority than Swappable processes. Most processes are type-swappable. They are the first candidates for swapping to disk and run at a Scheduler-derived priority based upon their past and current behavior, as well as their projected behavior and [user] assigned priority.

Processes that interact directly with the system devices via Assembler language I/I instructions must be resident, as are processes that utilize histogramming and/or the undocumented ?Spy System call. System Calls that demand process residency are detailed in the AOS Programmer's reference manual. As described in Chapter 1, too many resident processes can force AOS into memory "deadlock", requiring an Emergency Shutdown (Esd), - so System Manager's beware!

The purpose of declaring a process Preemptible is to give it a high process and memory priority. Unfortunately, there is no way to give a preemptible process a lower priority than one that is swappable; as can be done under recent releases of AOS/Vs.

For both Resident and Preemptible processes, should they go Compute-bound, they will tend to dominate the system, or at least lock out all the swappable processes; so again, beware!

Swappable processes are unique in that their performance is constantly being "monitored" by the process scheduler and their actual priority is altered based upon process behavior. More on this later.

Process Scheduling Basics

Under AOS, processes are scheduled for CPU time based upon what is known as their PNQF (Prior Enqueueing Factor). The process with the lowest PNQF which is also ready will be given control of the CPU for a specified period of time.

For Resident and Preemptible processes, the PNQF is none other than the user assigned priority 1 - 255. So in effect, the "father" controls the PNQF of these processes. For Swappable processes however, the procedure is quite different.

Because most processes are swappable, it makes sense to have the Process Scheduler compute the PNQF rather than to have it assigned by the user "himself". You see, having a static priority for a process throughout its life is not a very good idea because it fails to take into account what the process is actually doing. A high priority cpu-bound process could easily dominate the system, creating a system deadlock or some such problem; whereas, a low priority process predominantly I/O-bound, might never run at all.

The AOS solution to this problem is to allow processes to run on a time-slice, basis adjusting the process PNQF based upon its past behavior and other factors. The goal of this computation is to reward I/O-bound processes with a lower PNQF and to penalize compute-bound processes by increasing their PNQF. Thanks to an intimate relationship with the interrupt world, the Process Scheduler is capable of dynamically altering a process's PNQF as it gravitates towards I/O dependency or becomes more CPU-bound. AOS keeps track of process's behavior by adjusting its Timeslice Exponent (Tx).

For purposes of Scheduling evaluation, the Scheduler assumes a process to be in one of four states:

- Console I/O Bound (most desirable) Tx = 1
- I/O Bound - to non console-type devices
- Awaiting an IPC completion (other than to PMGR)
- Compute-bound (least desirable) Tx = 6

Time-Slices

Earlier, I mentioned that swappable processes are time-sliced, but I haven't really given a definition of what a time-slice is. Under most multi-process systems, a time-slice is simply an amount of cpu time given to a process for execution. If a process uses all the time allotted to it, fine. If not, the time-slice remaining is voluntarily relinquished and the process will have to wait for another slice of time in order to continue execution. For Resident and Preemptible processes, this definition is also accurate.

For Swappable Processes under AOS, this definition is modified somewhat. Here, a time-slice is simply a "yardstick" (or measuring device) used to more precisely determine a process's behavior.

Time-slices and Process Behavior

The AOS Scheduler will always give a process as much cpu time as it needs. How often the process is actually scheduled however is determined by its behavior, as I described earlier. In order to make the scheduling algorithm work smoothly, AOS utilizes the P.I.T. (Programmable Interval Timer) to allocate cpu time to a process. Unlike the Real Time Clock (RTC) which can interrupt only at fixed intervals (specified during AOSGEN), the P.I.T. can be "programmed" to interrupt at intervals as short as 100 microseconds and as long as 65.536 seconds.

Resident and Preemptible processes because of their priority are given a time-slice of 2.048 seconds. If they are still ready to run, then they will be given another 2.048 second slice of time until they all eventually block for some reason; which will relinquish control to the highest priority, ready, swappable process.

When a Swappable process first begins execution, AOS assumes it to be I/O bound and therefore allocates a small time-slice. Time-slices are doled out a sub-slice at a time. (A sub-slice being 32 ms.) At the end of a sub-slice, the tasks within the process are rescheduled.

If the process should use up its time-slice without blocking, the scheduler assumes the process is becoming more compute-bound, raising its PNQF and upping its time-slice amount to the next power of 2. Likewise, if a process blocks before using its time-slice, it is assumed to be heading more towards I/O dependency and is given a lower PNQF and a smaller time-slice.

If a process loses control of the CPU due to an interrupt and/or a higher priority ready process, before scheduling the new process AOS saves the remainder of the current process's time-slice so that the process can be properly restarted later.

The relationship between process behavior, priority and time-slice is illustrated in Diagrams 2-1a and 2-1b.

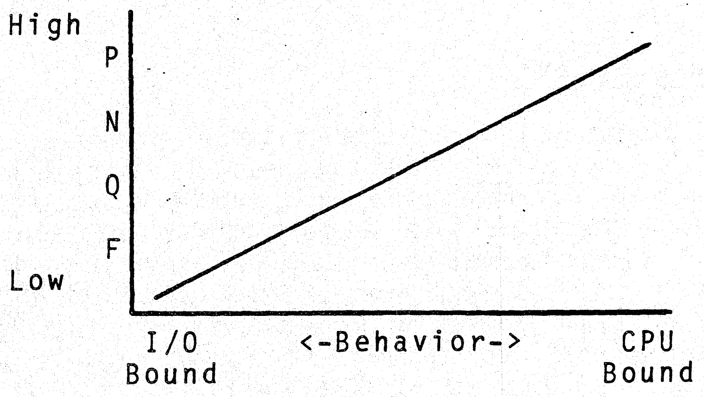


Diagram 2-1a.
Process Behavior vs. Process PNQF

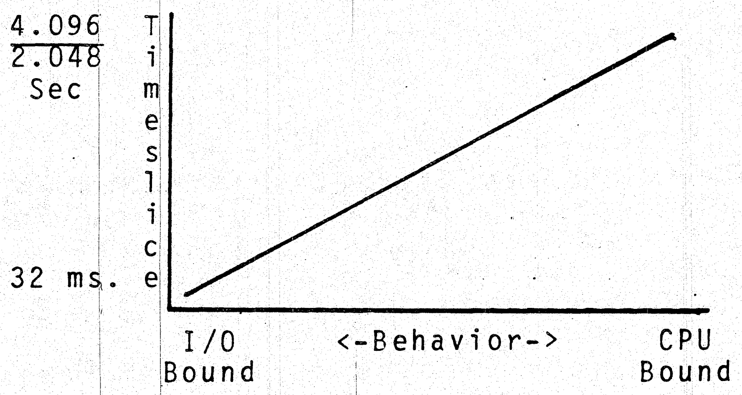


Diagram 2-1b.
Process Behavior vs. Time-slice Length

Time-Slice Computation

The actual time-slice value for a process is computed as:

$$Ts = 2^{Tx} * Ss$$

Where:

Tx is the Time-slice Exponent (1 - 6)
Ss is a fixed amount of time (32 ms.)

Each time a Swappable process becomes more compute-bound, the Tx is incremented by 1, allocating a larger time-slice. In case of a Priority 1 compute-bound process with a Tx of 6, the scheduler doubles the time-slice from a standard 2.048 seconds to 4.096. Remember however, that a time-slice is actually used a sub-slice at a time. Therefore, a swappable process will dominate the CPU for no more than 32 ms.

PNQF Computation

For Resident and Preemptible processes, the PNQF is simply the assigned priority for the process. For Swappable processes, the PNQF is computed via the following formula:

PNQF = 1BO + PCN + WPRI

Where:

PCN	is the Process Characteristic Number which is computed based upon values taken from Steady-State table (**PCNSS**), the Event-Count table (**PCNEC**), and the Reaction-Rate table (**PCNRR**) - otherwise known as the PCN Recalculation Tables.
WPRI	is the slot contents of the Weighted Priority Table. This table is indexed by the user-assigned process priority (1,2,3)

The computed PCN is that part of the PNQF which truly reflects the past behavior of the process. The PCN begins with the **S**teady-**S**tate value for a process's current behavior level and is incremented by the **R**eaction-**R**ate for that behavior level. The **E**vent-**C**ount table determines how long we will allow a process to remain at a given behavior level before being dropped to the next lower level.

It could be said that the PCN is a process's explanation for why it is now unblocked. Every time a process unblocks or runs out its time-slice, the PCN is recalculated.

Table 2-2 details the PCN Recalculation formulas, and Table 2-3 declares the contents of the PCN Recalculation tables.

Current PCN	New PCN
PCN < old PCN	The next lowest PCNSS value
PCN = old PCN	Old PCN
PCN > old PCN	Old PCN + PCNRR
PCN > old PCN and Ts. expired	The next highest PCNSS value

Table 2-2
PCN Recalculation

Unblocking Event Behavior Level	PCNSS	PCNEC	PCNRR
Console I/O	1	4	3
Non-Console I/O	13	1	3
IPC (non PMGR)	16.	2	3
Compute Bound	22.	1	3

Table 2-4
Recalculation Table System default values

Priority Weight Table (WPRI)

The Priority Weight Table is a table that allows the user-assigned priority to take precedence over the PCN calculation in the scheduling algorithm. This is done by giving the priority some "weight" in the PNQF computation. This table is indexed by the swappable process's assigned priority. As it is designed, this table simply contains the values 1,2,3 at the corresponding offsets. If the values in this table are made larger than values calculated from the PCN Recalculation tables, then the user-assigned priority can be made to dominate over the PCN calculations with regard to determining a process's actual PNQF.

The Ready Eligible Queue (RELQ)

Internal to AOS is a queue known as RELQ; the Ready, Eligible Queue. This queue maintained by the Core Manager and the Process Scheduler, is a linked list of all Ready/Eligible processes and is ordered by PNQF.

When it comes time to run a process, the AOS Process Scheduler starts at the front of RELQ looking for the first Ready process to be found. As the PNQF for a process changes, its position on RELQ will be shifted accordingly.

Altering the Process Scheduling Algorithm

Thus far we've examined how process scheduling is accomplished by AOS and how it relates to user assigned priority. It's also useful to be able to alter these algorithms from time to time. Unfortunately, for Resident and Preemptible processes there is nothing that can be done to affect the process scheduling outside of assigning different process priorities. A word of caution here: avoid running Resident and Preemptible processes at Priority 1. At priority 1 they will be competing with the PMGR for cpu time and that could negatively impact system performance; specifically with regard to console I/O.

For swappable process there are two kinds of changes that will affect the scheduling algorithm:

(1) Program changes altering the % of cpu time to I/O time

(2) Plugging the System Tuning Tables

"Fooling" the Scheduler

Through programming changes you can change the execution flow of the user programs and/or install routines that perform "dummy" I/O-type System Calls. Such calls can be made in order to "fool" the scheduler into "thinking" that the process is becoming more I/O bound, thereby lowering its PNQF. On faster CPU's this occurs naturally as a function of increased cpu speed.

As an experiment, I took a heavily compute-bound program and installed an I/O routine that was called at various points in the computational loops. This I/O routine simply requests AOS to read the same block of a 1 element shared file that had been open exclusively for this purpose. This I/O request indicates to the Scheduler that we are gravitating towards I/O dependency, when in fact this is not true at all. Because the request is to a frequently accessed shared file, no physical disk I/O is ever performed - the block being core-resident in a shared-page - the [so-called] I/O taking a negligible amount of time.

The characteristics of each program are usually significantly different. Frequently, a process's characteristics will have to do with what other processes are running in tandem at the time. In order to make this kind of technique work, some experimentation will be in order.

Plugging the System Tuning Tables

Patching the System Tuning Tables offers an excellent way to alter the Scheduling algorithm. A potential drawback to this method however is that it is officially not supported by Data General, even though it seems to work rather well.

A system tuning facility has been "wired" into AOS for years although it is undocumented and therefore probably unsupported by Data General. Currently, these tables can only be plugged using the System Tuning screen of **Aoswatcher** or by patching the System.Sy file (as mentioned above) with the Disk Editor, **DEDIT** (horror of horrors).

When **Deskwatcher** was being developed, the ?TUNE System Call (the key to System Tuning) was tested thoroughly and found to be workable with the WPRI table and the PCN Recalculation Tables. (There is another aspect of this call which doesn't seem to work and is therefore not currently used in **Deskwatcher** or **Aoswatcher**).

With **Deskwatcher/Aoswatcher** utilities, you can alter the above-mentioned tables as the needs of your system dictate. As the process load on the system varies throughout the day, you may wish to alter these parameters from time-to-time. If you find a setting that seems to work most of the time, you might want to "patch" these tables permanently in the .SY file and reinstall the system as necessary.

A word of warning is that patching the .SY file might leave you wide-open for unsupportability from Data General. It is best to check with your DG Systems Engineer to get a recommendation on how to do this and still insure supportability. Talk with a number of DG S/E's has indicated that this is probably not a big issue if the patch is properly applied. The table names to be patched are: **WPRI, PCNSS, PCNEN, PCNRR** respectively. They can all be addressed by those labels using the Disk Editor (DEDIT) and the .ST file produced during AOSGEN.

Plugging the Weighted Priority Table

The only valid reason I can think of for ever patching the Priority Weight Table is to force the user-assigned priorities (1,2,3) to take precedence over the behavior derived aspects of the scheduling algorithm. To accomplish this, you want to plug values into this table that exceed the largest value calculable with PCN Recalculation Tables. This largest value is computed from the compute-bound offset of these tables as:

Max-Value = PCNSS + (PCNRR * PCNEN)

On my AOS System, I have chosen: 100, 200, 300 for priorities 1,2,3 respectively.

Plugging the PCN Recalculation Tables

When plugging the PCN Recalculation Tables you must be careful to insure that the values plugged into each of the three tables exhibit a proper relationship with the figures in the other tables. To help you select the "appropriate" values to "plug" into the PCN Recalculation Tables, a few tips are in order.

As described in Table 2-3, process behavior can be broken into four groups (or levels if that makes more sense). The Steady-State values (**PCNSS**) declare the start of each behavior level, while the Event Counts (**PCNEC**) define the length of each group/level. The Reaction-Rate entries give the incremental value for each iteration of the Scheduling algorithm as a process is measured to be more compute-bound. This value declares how quickly we will drop to the next behavior level.

The most important thing to understand about these tables is that their values are all relative to each other; specific values have no meaning. In order to intelligently plug these tables permanently, you will need to first experiment these values using the System Tuning Screen of **Aoswatcher.**

To assist in this experiment, remember that the **PCNSS** value (for all but the Console I/O behavior level[1]) is computed as:

(**PCNEC** * **PCNRR**) + **PCNSS**
(values taken from the previous level)

A working example is probably in order here. I will use the Recalculation Tables from my own Desktop AOS system as an example of how these tables might be set.

This system is used predominantly in the following areas: 1) System software development; 2) Full CEO; 3) Manuscript writing and general word processing (using a non-DG Word-processor); 4) Various computer games from Pacman, Chess and Adventure to Startrek and a Stratego board of my own design.

Once the tuning screen was designed into my then "personal" copy of **Deskwatcher,** I saw instantly that the default values for the PCN Recalculation Tables (shown in Table 2-3) chosen by the designers of AOS were inadequate for a system being utilized in the way mine was. Some experimentation (and admittedly guess work as well), brought me to the values described in Table 2-4.

---

1) This should always begin with a value of 1; unless you wish to insert a new scheduling structure.

Unblocking Event Behavior Level	PCNSS	PCNEC	PCNRR
Console I/O	1	6	3
Non-Console I/O	23	4	3
IPC (non PMGR)	37	2	3
Compute Bound	45	10	3

Table 2-4
Recalculation Table working example
(All values in Octal)

From studying Table 2-4, several things should be evident:

1) Console I/O and Compute-bound processes are given the most and careful attention.

2) IPC-bound processes being more rare are therefore given only minimal attention. An Event-Count of 2 allows for one ?ISEND/?IREC sequence or a ?IS.R call to be executed before dropping to the next behavior level.

3) A diagramatical representation of the above table might be an hourglass figure that is flared at the bottom somewhat more than at the top. This means that we ask AOS to evaluate Interactive vs. Non-Interactive processes in a similar way, although Scheduling is a little more precise when dealing with non-interactive processes.

4) The Reaction Rate Table values are the same as the System Default. This makes plugging the Tables easier.

Summary

As you can see, Process Scheduling is a rather complex issue and you haven't heard the half of it. Looking into the internal structures of AOS you will run into Ghosts and Daemons and the Interrupt World, to name a few of the other components. Throw in multi-tasking and you really have a mess on your hands.

Unlike other areas of performance , altering the Scheduling Algorithm is one of the easiest ways to effect an improvement. Be ye' not afraid to experiment a little. You may be in for quite a surprise......

Chapter 3
The "GHOST" Process

The Ghost Context

Basic Theory

With most users, little is known about the Ghost context and yet it plays a vital role in matters of System Performance. Although minute detail is unnecessary, a rudimentary understanding of the Ghost will offer some valuable insight into the ways of AOS. For system programmers, knowledge of the Ghost can result in more effective programming; especially where I/O is concerned.

To begin with, the User Process is often referred to as the **Primary Context**; whereas the Ghost is frequently referred to as the **Secondary Context.** A good analogy concerning the Ghost is to see it as an alter-ego of the User Process instead of a separate process such as PMGR. The Ghost is wholly contained in GHOST.PR and GHOST.OL and lives in the root directory.

Basically, the Ghost is nothing more than a parallel extension of User Address Space sharing Task Control Blocks (TCB's) with the Primary Context when "it" is in control. An adten in shared-code This shared-code guarantees that there is only one physical copy of the code occupying memory; although there will be numerous "logical" copies present (one "logical" Ghost per process). This "initial" Ghost is created as a Secondary Context of Pid 2, OP:CLI
during final stages of booting the system (before the initial CLI prompt). Its code is loaded from GHOST.PR starting the process.

Due to the design of the hardware MAP, the maximum process logical address space is 32kw. Due to this construct, each process has 32kw available for each Context, Primary alogical" copies present (one "logical" Ghost per process). This "initial" Ghost is created as a Secondary Context of Pid 2, OP:CLI
during final stages of booting the system (before the initial CLI prompt). Its code is loaded from GHOST.PR starting the process.

Due to the design of the hardware MAP, the maximum process logical address space is 32kw. Due to this construct, each process has 32kw available for each Context, Primary aail

The Ghost has five main functions:

(1) An interface to/from the User Process

(2) A System Call Interface

(3) A PMGR Interface to handle Programmed I/O (PIO)

(4) Logic to handle the Debugger, Dedit & Sysdmp functions

(5) An interface to the System Scheduler for the Rescheduling of tasks within the Primary Context

Let's look at these functions in greater detail.

User <---> Ghost Interface

In order to off-load AOS, the Ghost pre-processes many of the system calls. This is the interface between the task-level of the User Processndle Programmed I/O (PIO)

(4) Logic to handle the Debugger, Dedit & Sysdmp functions

(5) An interface to the System Scheduler for the Rescheduling of tasks within the Primary Context

Let's look at these functions in greater detail.

User <---> Ghost Interface

In order to off-load AOS, the Ghost pre-processes many of the system calls. This is the interface between the task-level of the User Process and the Operating System. Regardless of which logic in the system actually processes the system calls, they all start off at the Ghost level through this interface.

Assumming that no pre-processing of the System Call is required, the Ghost simply handles the required "bookkeeping" and passes the request to the AOS Kernel. If no post-processing is required, the System Scheduler will eventually give control of the system back to the User Task to continue its operation. The User Task never talks directly to AOS, and AOS System Call never directly responds to or dispatches a User Task.

Ghost ---> System Call Interface

When a ?System call is placed by the User Process, control is first transferred to the Ghost Context. Some of these calls are in actuality Ghost calls, meaning that they are pre-processed by the Ghost before being queued to the AOS System Call Processor. ?Read/?Write and ?Proc/?Chain are examples of pre-processed calls.

There are several advantages to having the Ghost handle this pre-processing. One advantage is that the AOS Kernel is off-loaded and the User Process is more accurately charged for the CPU time it utilizes. Another advantage is that because the Ghost Context is an alter-ego of the User Context, it [potentially] has easier access to the User Program address space facilitating page sharing in order to move data to/from the Ghost Buffers and user space, as well as maintaining the user Task Control Blocks.

An example of this pre-processing occurs in a data-sensitive read from the disk. Let's assume the Primary Context issues a ?Read system call requesting 200 bytes of data from an open file named Zippy. The Ghost will intercept the call and request AOS to read the [physical] disk block (512 bytes which contains our requested 200 bytes) into one of the Ghost buffers in unshared memory. After the read is finished, Ghost unpacks the requested 200 bytes into user address space. The AOS Kernel sees only the start of the direct block I/O request into the Ghost buffer.

Ghost <---> PMGR Interface

To simplify PIO transfers (which are already slow enough as it is), the Ghost has a direct line to PMGR via "secret" IPC ports. For the most part, the AOS Kernel is not involved in the data transfer to/from the console-type devices. To enhance things further, PMGR has available two undocumented System Calls: **?MBTU** and **?MBFU**. This allows console data to be passed directly between PMGR and the Primary Context without Ghost intervention. (To use ?MBFU/?MBTU a process must have the PMGR Privilege)

In a way similar to the Ghost ---> AOS interface, the Ghost handles pre-processing and post-processing of ?Read and ?Write System Calls while PMGR (in conjunction with the AOS Interrupt World) handles the actual I/O transfers.

One of the obvious reasons that PIO devices are slower than data channel devices (aside from the data channel speed) is because of this IPC protocol between PMGR and the Ghost.

The main reason why IPC privilege is not recommended for users is because they could then attempt bogus IPC calls to PMGR's (undocumented) control ports thereby crashing the system or triggering unusual or flaky PIO device operation.

Ghost ---> System Scheduler Interface

The Ghost has the option of requesting task rescheduling for the Primary or Secondary Contexts. It invokes this privilege mainly after post-processing System calls. Without this mechanism, post-processing of System Calls would have the liability of violating the AOS scheduling algorithm; thereby defeating a portion of the Ghost's purpose. This could unfavorably impact System Performance.

Debug - Dedit - Sysdmp Logic

Because the Debugger and Disk Editor (Dedit) are often required by many users simultaneously, they have been written into the Ghost libraries making them common to all user processes. And, if you take a closer look, these utilities all share the same command and operation format making the Ghost the obvious choice for the placement of this code.

Ghost Summary

So as you can see, the Ghost does indeed play an important (albeit invisible) role in matters of System Performance. (After all, why do think we call it the **Ghost**?) Although this role is an indirect one, it is nevertheless important and should not be overlooked.

And of course, the presence of the Ghost tends to explain where some of the memory pages are "disappearing" to. More information on locating "lost" memory pages is described in Chapter 1 - Memory Management.

Happy H~~a~~unting!.....

```
*********************************************
*                                           *
*                                           *
*              Chapter   4                  *
*            The Disk World                 *
*                                           *
*                                           *
*********************************************
```

Chapter 4
The Disk World

Disk World Performance Guidelines

When it comes to System Performance, disk seek time is always the problem. Luckily, there are a number of things that we can do to remedy the situation. This includes: careful file placement, effective directory utilization, choosing filenames and hash frame sizes carefully, and even being conscious of the importance of declaring proper searchlists. Each idea by itself may not generate all that spectacular of results, but used in conjunction with one another can be very powerful. Let's take a look what these methods and facilities can do for us.

Drives and Controllers

When configuring multiple disks on the system, you need to look at the issue of multiple disks and/or multiple controllers. Frequently, when a shop attempts to resolve their disk utilization problems, the only solution considered is to add additional disks to the system. All too often however, this does not fully produce the expected results; usually for one simple reason, controller interference becomes so heavy that the advantage of an additional disk (or disks) is somewhat negated.

Controller interference is where I/O to a selected disk drive must be queued to the I/O world (ie. stalled) because the controller is currently handling an I/O request to another drive on the same controller. While we can usually perform concurrent seeks on a controller, only one block transfer may occur at a time. This is usually because the controller chip(s) contain only one set of sector buffers. Therefore, file placement becomes quite a crucial issue.

Although often relatively expensive (compared to disk drives), adding additional controllers to a system can dramatically improve system performance. Many installations needing only two disk drives often place each on a separate controller in order to maximize I/O speed. If speed (and not drive capacity) is the issue, it makes sense to invest in faster disks/controllers even though they may be limited in capacity. To get both speed and capacity, you must pay the price. Using the Dstats screen of **Aoswatcher** will help you determine whether adding another controller is needed or (after having been installed) was worth the investment.

File Placement Guidelines

Aside from adding another disk drive to the system (if not already done), file placement becomes a very important issue. The idea of file placement is to physically place commonly accessed files as close together as possible, thereby minimizing drive seek time. This should be done starting with system installation; waiting until later will only complicate matters.

For example, on my desktop system I loaded all of my writing and programming material onto the disk only after the disk had been reorganized. You see, after the initial system installation, files on the disk are not in what I call "user-oriented" order. Once the System Installation was complete and tested, I dumped the directories and files in "proper" order onto a tape in order that the disk could be [re-]organized into file access order; thereby improving performance when loaded back onto the disk.

Essentially what was required was to perform two system installations; except that in the second installation I loaded the files from my backup tape rather than file 7 of the release tape. The advantage in doing this is that the disk is now loaded in my order, not in release tape order. Some time was required to plan this selective dump, but it was well worth the effort in terms of improved System Performance.

In general, files that are never modified (such as .Pr and .OL files) should be loaded near the front of the disk and constantly changing files (such as word processing text files and fluctuating databases) should be placed near the [so-called] end of the disk. Placing potentially updatable files near the beginning of the disk forces them to be spread out towards the end of the disk as the additions are made to it over time. This forces rather lengthy disk seeks when bouncing from the front to the back of the file and vice versa. Application programs referencing these files will suffer greatly and will impact all other processing as well due to the massive disk seeks required to access the newly added blocks.

Directories and CPD's

Although file organization can improve System Performance (not too mention making it easier to find things), I see so many disks in which files are just scattered about. Proper disk organization is essential; even more so on the Desktop when you consider that we are working with a "crippled" disk to begin with. If your disks are poorly organized, you won't really notice how bad things are until after you put some order into the chaos. At least then you'll have organized chaos.

Under AOS, directories and control-point directories (herein called CPD's) are the key to file organization. To fully utilize directories and CPD's we need to draw a distinction between these two aspects of the same thing. (Say what?)

Essentially, directories and CPD's function the same except that CPD's allow you to restrict the amount of space used by a particular group of files. The CPD max-size is simply a ceiling on the amount of space available to that file group. AOS does not actually allocate that amount of disk space (as does RDOS with its equivalent Partitions). Other than the CPD max-size, the only other real advantage to a CPD that comes to mind is the ability to determine the space used/remaining for a given file group. This allows you to get an "eye" for how much space a given group of files is actually "costing" the disk.

Directories incur overhead in the form of invisible disk space taken up by **D**irectory **D**ata **B**locks (**DDB**'s). DDB's house a number of "invisible" directory structures known as **D**irectory **D**ata **E**lements (**DDE**'s). There are 5 kinds of DDE's: **F**ile **I**nformation **B**locks (**FIB**'s), **F**ile **N**ame **B**locks (**FNB**'s), **F**ile **L**ink **B**locks (**FLB**'s), **F**ile **U**ser **D**ata areas (**FUD**'s) and **F**ile **A**ccess **C**ontrol blocks (**FAC**'s). These internal structures will be fully explored later in this chapter.

General Disk Overhead

Having discussed directory basics, let's put this stuff to use by talking about disk organization in general and what it can accomplish towards improving system performance.

Most AOS disks I have seen are usually a mess organizationally. One only has to do a Filestatus on the Root directory or :UTIL to bear that one out. When I look in the Root directory of most systems, I find an incredible morass of files; everything from .PR's and .SR's to backup files and people's personal letter texts. On one system I looked at, the Root directory would have been a gold mine for the local gossip columnist.

Directories are about file classification. Granted there is some overhead associated with using directory structures, but the organization afforded is well worth the investment of time and space. The major directories on the system that are candidate for re-organization are the Root and :UTIL.

For example, on my system we created a directory :MACROS and moved all system related .CLI files there. This helped to clean up :UTIL and the CEO directories considerably. Likewise, I organized most of the files in :UTIL into sub-directories to make things easier to find. As for the Root directory, I moved files such as +.St onto a diskette LDU as they aren't normally needed unless you are installing program patches.

As we shall see later, directory organization makes for faster filename look-up due to the nature of the hashing algorithm. I can't encourage directory organization enough; and yet, you will probably ignore this advice completely. I hope not.

Shrinking Directory-space Overhead

A characteristic of the AOS internal directory structure is that it grows (if necessary) but it never shrinks. Like disk fragmentation, DDB's can also fragment as files are added and deleted; in much the same way as the overall disk space. The hash frame size (as we shall see) can affect directory space utilization to some degree but is not a cure-all; especially in an "active" directory.

Just like the disk drives, directories need their space compressed as well. There are two ways to accomplish this:

- 1 - Fixup

- 2 - Manual Compression

FIXUP

The FIXUP utilities support the shrink-directory-files feature. This feature de-allocates DDB's that no longer contain active DDE's; the net effect being lesser directory overhead. Until AOS release 7.0, this was a selectable option. Currently, it is no longer selectable. At times, I find this undesirable.
To remedy this situation I keep a copy of Release 6 FIXUP resident in :UTIL for the times when FIXUP becomes necessary (due to a system crash) and I want to keep the unused directory space allocated. Should the system crash while I am in the process of reloading directory structures, I want the allocated directory space to remain in order that it be reused when I restart the Load process.

If all disks are declared by DFMTR as system disks (containing at least a minimum AOS configuration), you can shrink both disks by booting each disk alternately, running FIXUP on the "other" drives.

Compressing Directories Manually

On an individual basis, directories can be compressed manually via the CLI. Although not a perfect solution, manual compression will not only free up directory space, but it also increases the chance of the Directory Data Blocks being localized in a contiguous area. At this time, it might also be a good idea to check the appropriateness of the hash frame size for the directory. Manual compression is best accomplished via DUMP/LOAD or MOVE; although the MOVE option does have some potential side-effects.

The formula for manual compression is to Dump or Move the files out of the target directory, delete the directory (freeing up all the allocated Directory Data Blocks), recreate the directory structure, and finally, Move or Load all the files back into the directory.

Here's an example:

```
     DIR  Pathname_of_Target_Directory

     DUMP @MTC0:0 #
        -Or-
     MOVE Somewhere_Else
------
     DIR  Parent_Directory
     DEL  Target_Directory
     CRE/DIR/HASH=nn/MAXS=nnnn   Target_Directory
------
     DIR Target_Directory

     LOAD @MTC0:0
        -Or-
     DIR Somewhere_else_Directory

     MOVE Pathname_of_Target_Directory
```

The one possible drawback of using the move command is in moving the target directory files somewhere else on the same disk. In the process of reducing DDB overhead for a single directory, you may force fragmentation on the disk as a whole. Use the MOVE command only to another drive (which may force fragmentation there instead). Or, if you have a desktop system, create a diskette LDU and temporarily move the files onto it. A diskette can also be used as a sequential dump device; just like diskettes 4 - 17 of the AOS release media.

If the directory is frequently accessed, you can use **Aos-watcher** to check the average seek distance for any noticeable improvement in performance. Otherwise, use the **Disktrace** to locate the DDB's. The DDB's will of course graphically disclose the results.

A Third Possibility

Now there is a third method useful for compressing directory space, which is known as disk compression. Although more time consuming than the methods described sp far, it is a far more precise mechanism. This is discussed later on in this chapter.

A side note here is that when doing a compression dump, if the purpose is to shrink the directory files, be sure to dump the contents of the directories, recreating the directory structure, rather than dumping the directories themselves.

Hash Frame Size

Unlike data files, a directory does not have a file element-size per se'. Instead, we use what is known as the **H**ash **F**rame **S**ize to control directory efficiency, as well as theas it's efficiency. Like database structures, hash values should be prime #'s in order to fully utilize the disk space, and like database structures, filename hashing can rD being a kind of database. **Hash Frame Size** is the value that determines the initial (and subsequent) disk space requirements of a directory or CPD as well as it's efficiency. Like database structures, hash values should be prime #'s in order to fully utilize the disk space, and like database structures, filename hashing can result in synonym problems: having a filename resolve to the same "home" location along with numerous other filenames.

How the Hashing Algorithm Works

Filename "hashing" simply consists of adding up the ascii representation of all the characters in a given filename, dividing by the **H**ash **F**rame **S**ize, taking the remainder from that division and using it as a block offset in the **D**irectory **D**ata **B**locks (**DDB**'s). Diagram 4-1 illustrates this algorithm. Once a DDB is found, a scan of that block is made looking for an FNB that matches the filename. If the FNB is not found, AOS scans the **D**irectory **O**verflow **B**locks until the FNB is either found or the chain of **D**irectory **O**verflow **B**locks for that hash value is exhausted. As we shall see later, once located, the FNB points us to the FIB, and that points to the file disk blocks themselves. Later, we'll discuss the System Performance implications in choosing a proper hash value.

```
"FILENAME" with a Hash Frame Size of 7 computes as:

          F  <106>
          I  <111>
          L  <114>
          E  <105>
          N  <116>
          A  <101>
          M  <115>
          E  <105>
     _____________
     7 |  1105      remainder=3  ¦  Home Location is Block #3
       |_________
          122       (ignored)
```

Diagram 4-1
Hashing Algorithm Example

Large vs. Small Frame Sizes

Although a small **Hash Frame Size** is economical in terms of disk space, if the directory has a large number of files in it, System Performance can suffer. This is because there are fewer directory blocks in which to hash a given filename amongst. Therefore, the chances of overflowing a given filename's home location (block) are much greater, resulting in **Directory Overflow Blocks**. These overflow blocks will of course [potentially] require additional disk accesses to locate a given **File Name Block (FNB)** - See Diagram 4-2a. If too large a **Hash Frame Size** is chosen, there will be little or no **Directory Overflow Blocks**, and many of the allocated directory blocks will be empty.

A large **Hash Frame Size** may improve System Performance somewhat, but at the expense of wasted disk space. By comparing Diagrams 4-2a and 4-2b, the contrast between Large and Small **Hash Frame Size** should become readily apparent.

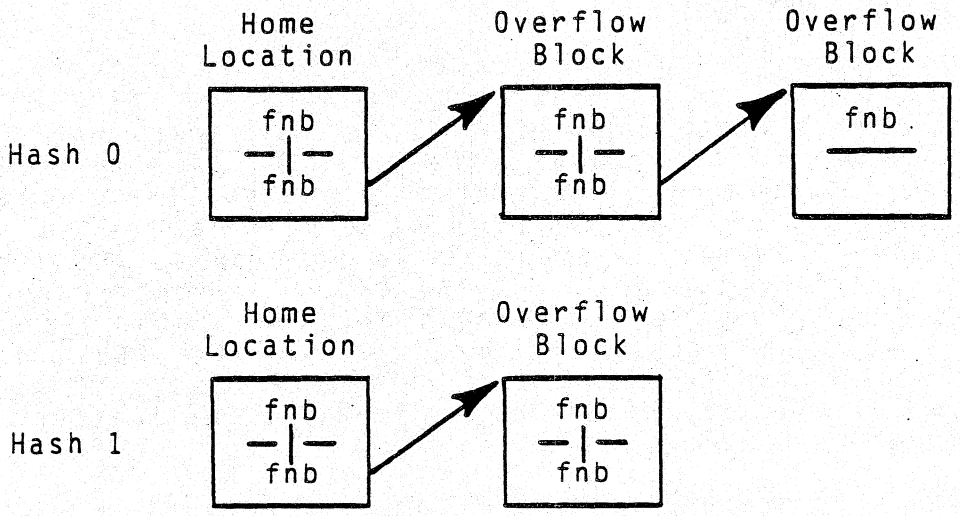


Diagram 4-2a
Hash Frame Size too Small

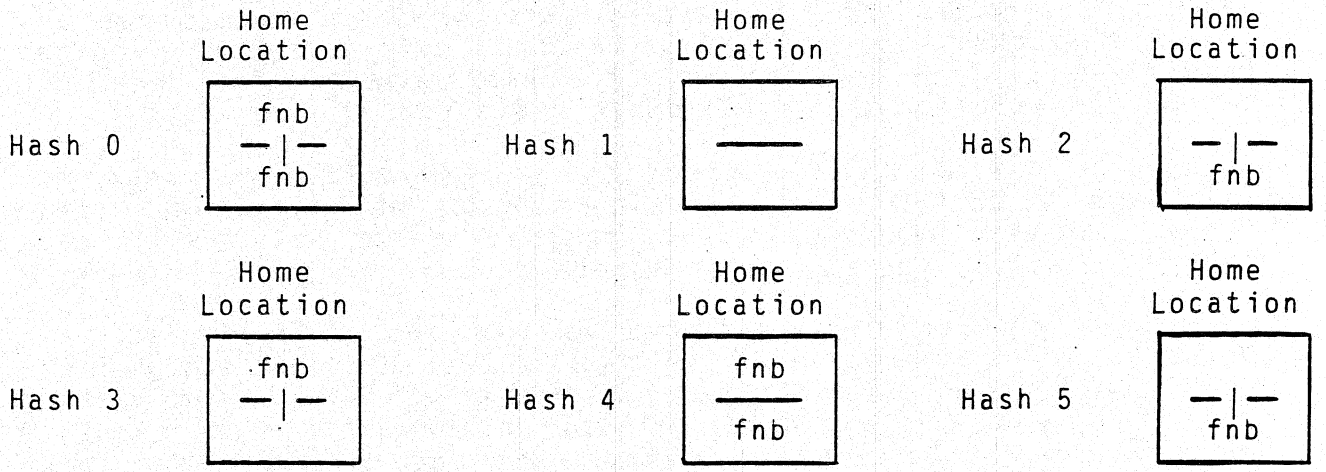


Diagram 4-2b
Hash Frame Size too Large

A good rule of thumb for choosing a directory's Hash Frame Size is the following formula:

H.F.S. = (# of Files in Directory) / 20

This formula attempts to encourage a rather even distribution of filenames amongst the Directory Data Blocks, although it may not always be the optimum formula. Filenames can affect system performance. They are not necessarily a trivial concern.

At any rate, as you can see, directories and CPD's are important to System Performance, and the Hash Frame Size is the key to insuring that they function in the most efficient manner possible.

Filenames and Filename Length

Suprisingly, proper choice of disk filenames can play an significant role in the improvement of system performance. However, because meaningful filenames also serve as documentation, we never consider the performance impact of choosing such filenames.

To begin with, filenames (with any extension) optimally should be less than 12 characters in length. This is because the filename will fit completely within one File Name Block (FNB), reducing directory overhead.

The importance of filename choice is of course due to the nature of the hashing algorithm itself. Axiom 10 of Fitchmouse's Laws of Computer Dynamics states: "Chosen filenames will always hash to the same "home" Directory Data Block forcing directory overflow blocks to be created". In other words, filename choices are frequently not in line with the hashing algorithm resulting in DDB's that overflow, while other DDB's remain empty.

With the Disktrace facility of **Aoswatcher** you can run filenames through the hashing algorithm to determine what their "home" location will be. If too many filenames hash to the same "home" location, you may elect to change some of the filenames.

On a particular system I helped design, we used this method. As it turned out, the hash value we chose was perfect with regards to DDB efficiency and poor as far as the hashing algorithm went. After installing the database files and such, we noticed that 3 of the DDB's had a string of overflow blocks "attached", while a number of the remaining DDB's were still empty. Because the user's never accessed files by their name (the software did that), our solution was to rename the files giving them names like: FILE012, FILE224, FILE969 etc. Although the names were not as easily recognizable as they were before, we did end up with the most efficient structure possible.

The effects of File Element-size

File element-size can significantly affect system performance and yet is a concept not understood well by the average user. To alleviate the problem of users not understanding its effects, I usually include file element changes in macros so that users will not have to be concerned about this important feature while at the same time being guaranteed that it is properly implemented.

A file's element size is simply the number of contiguous disk blocks that will be allocated for the file each time it requests additional space. This space is technically called a **F**ile **D**ata **E**lement. Choosing the appropriate element size is important. You should be aware that file space allocation is a time consuming task; especially on a large disk that is relatively full.

Therefore, if a chosen file element-size is too small, System Performance degradation will result. This is caused by the system needing to frequently allocate small amounts of disk space as the file increases in size.

Likewise, if the element-size is too big, unnecessary disk space will be wasted and this is not reported in the Filestatus command. So if you are having trouble in getting file sizes to add up to the [reportedly] used CPD space, it is probably due to allocated and yet unused disk blocks from files with too large an element-size.

Earlier, I discussed the importance of directory organization. Now I want to go one step further and talk more about file element-sizes with regards to Macros and HELP files. These two kinds of files are often less than 4 blocks in length and yet this is the file element-size usually chosen (if for no other reason, than by default). On my system, because there are hundreds of HELP files and macros (each wasting 2 - 3 blocks from too large a file element-size), until I shrunk the element-size to be in accord with the particular file, we were wasting over 700 disk blocks. To resolve this problem I wrote a macro to convert file element-sizes. Although it is rather complex, the essential statements appear below:

```
RENAME Filename<,.Xyzzy>
CREATE/Type=type-of-Xyzzy-file/Elem=nn Filename
COPY/A Filename<,.Xyzzy>
DELETE Filename.Xyzzy
```

Forcing the Element-Size

As a reminder, the Cli and programs utilizing the ?Create System Call default the element-size to 1. This of course can be quite a problem when using utilities (such as word processors and compilers) that don't allow you to declare a file's element-size. A simple solution to this (although it may not work with all utilities) is to create the file via Cli specifying the element-size before invoking the utility program. The program will then use the existing file rather than creating a new one. Additionally, some utilities (such as Masm) have a switch to allow a larger File-Data-Element on its created files.
An example might look something like:

```
CREATE/ELEMENTSIZE=32  PROG_LIST_FILE
LIST    PROG_LIST_FILE
XEQ/L   USER_PROG
LIST/G
QPR     PROG_LIST_FILE
```

Of course the above commands could be made into a macro for the sake of simplicity.

A working example of the effects of file element-size is detailed in the example below. To create this example we ran a personality analysis program several times varying the element-size of the output file from 1 to 32. The results are rather impressive:

Element size	User I/o's	Allocation I/o's	Total I/o's	Allocation Overhead
1	5860	5860	11720	100 %
4	5860	2930	8790	50 %
16	5860	733	6600	12.5 %
32	5860	367	6230	6.2 %

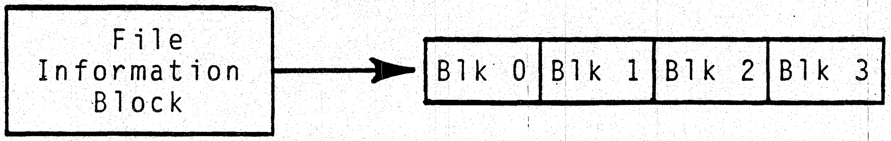
As you can see from the above figures, a file's element-size figures in dramatically. In order to eliminate any variables interfering with the measurements being taken, we modified the program so that at the end of each run rather than terminating, it would loop back to the beginning and wait for operator intervention. This eliminated the possibility of program load I/o interfering with the measurements. Also, because AOS buffers disk data (meaning no actual disk I/O is performed if the block is currently core-resident), we used four identical copies of the same file, opening them prior to the main execution path of the program.

File Index Levels

In addition to the FDE, overhead is incurred from the index blocks generated for a file larger than one File Data Element in length. For files with more than one FDE, a Random Index Block (RIB) structure is created by AOS. These RIBS are used to point to FDE's (for the first 128 FDE's) or to another level of RIB which points to 128 FDE's. AOS restricts us to a maximum of 2 index levels which allows a maximum of 16,384 FDE's per file. Each pointer in a RIB is 2 words long allowing a maximum of 128 RIB/FDE pointers per Random Index Block.

Although these index blocks are invisible to the user, they do take up disk space and can slow the file access considerably. Let's look at a couple of examples of what a file with and without RIB structure might look like. In these examples, assume a File Element Size of 4. The File Information Block is where the Filestatus information is kept.

One FDE Present



1 - 128 FDE's Present

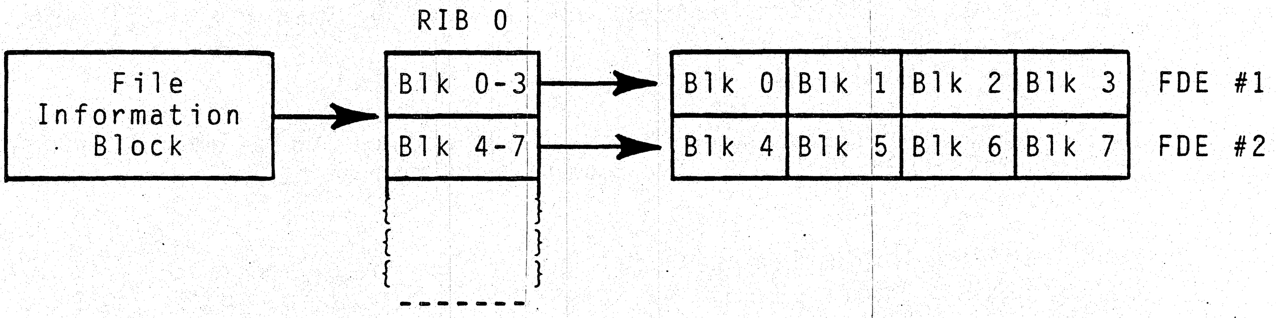


Diagram 4-3
Effects of File-Element Size

The LDU Bitmap

The Bitmap is an invisible "file" structure that exists on each and every disk-type device on the system. It is created by the DFMTR utility. Essentially, the Bitmap is used to keep track of all the free and used disk blocks on a given physical disk unit. The Bitmap keeps track of only the "visible space" on an LDU; ie. the logical disk addresses. The initial 10 blocks on each physical unit are always assummed to be in use and are therefore never recorded in the Bitmap.

With the **Aoswatcher** utility, we can examine the bitmap of each LDU and quickly determine whether or not fragmentation exists. What you are looking for is large groups of '1' Bits or '0' Bits. This indicates little to no fragmentation.

The following segment from a bitmap display is a good example of a disk with little fragmentation:

```
2000 177777 177777 177777 177777 177777 177777 177777 177777
***
6000 000000 000000 000000 000000 000000 000000 000000 000000
```

A segment from the same area on a fragmented disk might look like:

```
2000 176534 055360 146357 055555 177777 177777 106573 177777

2010 056543 177777 177777 000000 176534 036253 177777 000000
```

In the above examples, each group of octal digits represents 16 disk blocks, with each octal digit in a number group representing 3 disk blocks (excepting Bit 0). The rule of thumb is: if the bit corresponding to a disk block is a zero, then the block is available for use; if it is a one, then the block is in use.

For example, if the number group in a Bitmap display were:

136527 (1 011 110 101 010 111 in Binary)

Then blocks 0,2,3,4,5,7,9,11,13,14,15 of this group are in use and blocks 1,6,8,10,12 are available for use.

The Bitmap occupies several disk blocks based upon the size of the disk. Each block of the Bitmap represents 4096 disk blocks on the LDU. Diagram 4-3 compares the Bitmap size to the total disk size for the 3 available disk sizes.

Disk Size	# of Blocks	Bitmap Size (Blocks)
15mb	35968	8
38mb	74767	19
70mb	135904	35

Example of Relative Bitmap Sizes

Finding the Bitmap

Using the **Aoswatcher** utility, it is easy to find the Bitmap for a given disk. The first step is of course to open the disk unit for block display. The Bitmap is pointed to by the **D**isk **I**nformation **B**lock (Physical Block 3) as shown in Diagram 4-4. Details on this block appear in the upcoming section on Disk Tracing.

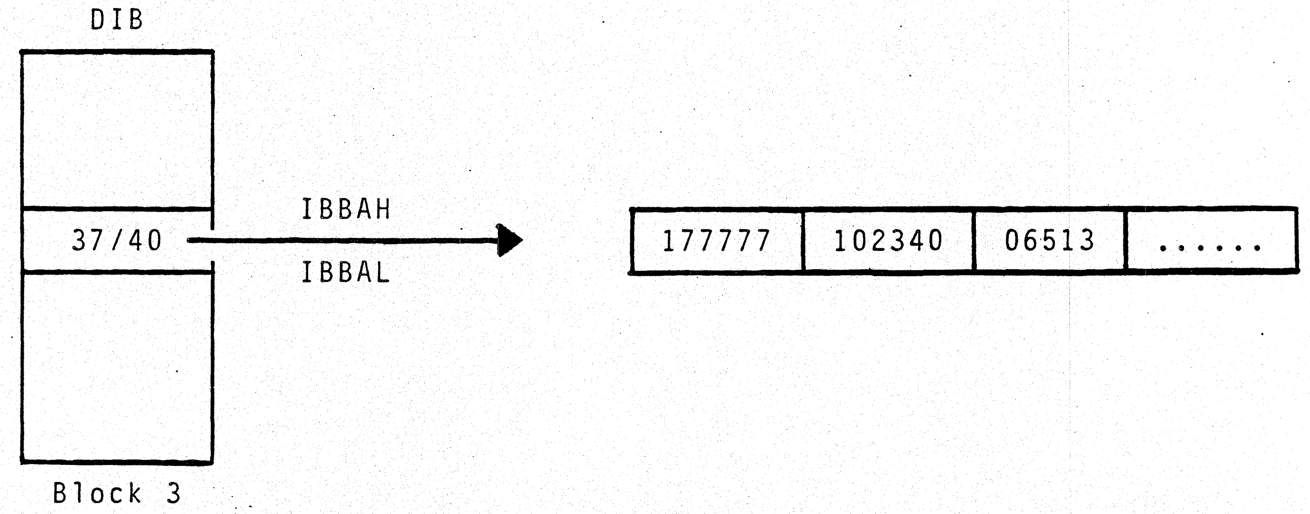


Diagram 4-4
Locating the Bitmap

To find the Bitmap, first display the DIB using the disk-trace command: BL 3. At offset 0037 (IBBAH/IBBAL) is a 2-Word logical block address of the first block of the Bitmap. Blocks of the Bitmap are contiguous so it is easy to locate the remaining segments.

Disk Compression

As files are created, deleted and modified, the disk will begin to fragment. The greater the activity of this kind, the quicker fragmentation will begin to occur. The main indicator of disk fragmentation is that access times begin to increase and the performance of the machine slowly degrades. Quite frequently, the gradient is so slight as to not be noticed. You simply settle in to the new (slightly slower) speed and are unaware of the problem until the response time really gets drastic; then it's already too late - something must be done!

AOS doesn't support any form of disk compression per se', and for good reason. The internal disk structure (as we shall see) is way too complex to be "squashed" in a safe manner, so it must be done via a Dump/Load. Although this requires careful planning and a decent amount of time to accomplish, it is well worth the investment.

As time goes by, it is probably a wise thing to dump and reload the disk(s) in directory/access sequence, effecting compression. The time spent periodically doing this will be repaid with a smoother, faster running system. How often to do this is based purely on the amount of file creating, deleting and appending being done on a given disk LDU. Frequent file additions will dictate that the disk be compressed more frequently.

Disk Fragmentation - Disk Compression

In the following diagrams you will be able to watch the BITMAP of an LDU as blocks are added, deleted and compressed. From these diagrams, the problems caused by disk fragmentation should become evident.

```
Physical Block: 000412
000   177777 177777 177777 177777 177777 177777 177777 177777
010   000000 000000 000000 000000 000000 000000 000000 000000
020   000040 000000 000000 000000 000000 000000 000000 000000
030   000000 000000 000000 000000 000000 000000 000000 000000
*****
060   177777 177777 177777 177777 177777 177777 177777 177777
*****
370   177777 177777 177777 177777 177777 177777 177777 177777
```

Diagram 4-5a
Blank Disk - Except Initial Files

This is the way the bitmap normally looks after you format a disk and install the initial system files. Notice that virtually no block fragmentation exists.

```
Physical Block: 000412
000   177777 177777 177777 177777 177777 177777 177777 177777
010   173777 177777 177777 177777 177777 177777 177777 177777
020   177777 177777 177777 177777 177777 177777 177777 177777
030   177774 000000 000000 000000 000000 000000 000000 000000
040   000000 000000 000000 000000 000000 000000 000000 000000
050   000000 000000 000000 000000 000377 177777 177777 177777
060   177777 177777 177777 177777 177777 177777 177777 177777
*****
370   177777 177777 177777 177777 177777 177777 177777 177777
```

Diagram 4-5b
Disk after Files have been Loaded

After files have been loaded, notice that we still have virtually no disk fragmentation. AOS obtains disk blocks on a per-cylinder basis when ever possible, loading the disk from front (low block addresses) to back (high block addresses). For comparison, running with this disk gives us an average seek distance of 49 Cylinders.

```
Physical Block: 000412
000   177777 177777 177777 177777 177777 177777 177777 177777
010   176020 141747 177740 000004 007770 003775 004000 002002
020   103677 177741 160000 000760 101777 177777 177763 160767
030   137777 101776 007777 173774 005770 077607 177700 034037
040   003777 100077 177777 177746 177777 177757 177770 000000
050   037607 140060 177377 004077 102377 177777 177777 177777
060   177777 177777 177777 177777 177777 177777 177777 177777
*****
370   177777 177777 177777 177777 177777 177777 177777 177777
```

Diagram 4-5c
Disk after Files have been Added and Deleted

As files are added, appended to and deleted, the disk begins to fragment. Obtaining contiguous space (for large FDE's) becomes somewhat difficult now. As a file is appended, it's blocks will physically exist all over the disk. This of course increases disk seek time, impacting System Performance. Running with this disk gives us an average seek distance of 85 Cylinders.

```
Physical Block: 000412
000   177777 177777 177777 177777 177777 177777 177777 177777
010   176777 177777 177777 177777 177777 177777 177777 177777
020   177777 177777 177777 177777 177777 177777 177777 177777
030   177777 177777 177777 177777 177777 177777 177777 177777
040   002000 100740 000044 000004 140000 000050 000000 000000
050   000200 040020 002200 004010 003377 177777 177777 177777
060   177777 177777 177777 177777 177777 177777 177777 177777
*****
370   177777 177777 177777 177777 177777 177777 177777 177777
```

Diagram 4-5d
More Files Added - Fragmentation is somewhat Hidden

As we place greater storage demands on the disk, AOS will fill up the fragmented "holes" in LDU, making it appear as though there is no fragmentation at all. Of course, the performance will continue to degrade as the average seek distance climbs higher and higher. Running with this disk gives us an average seek distance of 109 Cylinders.

```
Physical Block: 00412
000   177777 177777 177777 177777 177777 177777 177777 177777
010   000000 000000 000000 000000 000000 000000 000000 000000
020   000040 000000 000000 000000 000000 000000 000000 000000
030   000000 000000 000000 000000 000000 000000 000000 000000
*****
060   177777 177777 177777 177777 177777 177777 177777 177777
*****
370   177777 177777 177777 177777 177777 177777 177777 177777
```

Diagram 4-5e
Disk Unloaded - Most Blocks Freed

Unloading the Disk in File order frees up the blocks and of course improves the average seek distance significantly. Running with this disk again gives us an average seek distance of 49 Cylinders.

```
Physical Block: 000412
000   177777 177777 177777 177777 177777 177777 177777 177777
010   173777 177777 177777 177777 177777 177777 177777 177777
020   177777 177777 177777 177777 177777 177777 177777 177777
030   177774 000000 000000 000000 000000 000000 000000 000000
040   000000 000000 000000 000000 000000 000000 000000 000000
050   000000 000000 000000 000000 000377 177777 177777 177777
060   177777 177777 177777 177777 177777 177777 177777 177777
*****
370   177777 177777 177777 177777 177777 177777 177777 177777
```

Diagram 4-5f
Files Have been Reloaded - Blocks Compressed

Based on the above diagrams, it is worth reiterating that over a period of time Disk Fragmentaion becomes somewhat "hidden". Displaying the Bitmap no longer brings it to light. To detect the fragmentation you will need to measure other areas such as process performance and the average seek distance on the disk. Running with this disk gives us an average seek distance of 63 Cylinders; rather impressive compared to 85 and 109.

Searchlists and Performance

Searchlists can dramatically impact system performance and therefore should be examined closely. Because searchlists can be defined by any user, it is possible for that user to contribute to System Performance improvement. Likewise, careless use of searchlists by that same user can cause unfavorable impact on the system.

There are two things to keep in mind concerning searchlists:

- If possible, order directories in your searchlists so that the directories with the most frequently accessed files occur first in the searchlist.

- Keep the number of directories on your searchlist to a minimum.

Searchlist should rarely have more than 4 directories on them, and even that is excessive. (Of course, as I write this, my searchlist has exactly 4 directories on it.) Through the use of macros, searchlists can be pushed along with the Cli Environment, set to your specific needs at the moment and then restored via a Cli POP command. Another point to note is that the Peripheral Directory (:Per) never needs to be on a searchlist because it files are easily referenced via the "@" prefix; such as: @Mtc0, @Con1, @Lpt etc.

Searchlists are sweet and simple, yet they can be deadly if they are mis-used.

Making Use of :PER

For the most part people seem to know very little about the :PER directory other than the fact that it is a necessity. I call :PER a transient directory because "files" tend to come and go. Typically, they last as long as the current system remains up and running.

:PER Basics

:PER is a directory designed to house only device entries and IPC files Although technically speaking any file can live in this directory, unnecessary files will significantly slow the file search algorithm. For the most part, the peripheral directory is intended for System use only - keep user related files out! The only exception to this rule - IPC Files - will be discussed later.

When AOS needs to locate a device, it is a simple matter to locate :PER and scan the **D**irectory **D**ata **B**locks for the FNB. Devices are written into :PER during the initial system boot stages, while the Queue entries are created later by EXEC. An entry is created for every controller configured into the system via AOSGEN. Because the system has no real way of knowing the number of devices present on a controller, a device is written into :PER for every possible possible device on that controller, whether it physically exists or not. This ends up putting unnecessary file names in :PER, which I order deleted during execution of the UP macro. For example, I've gen'ed in MTC and MTC1 controllers for my Desktop. At boot time, AOS writes device names for MTC0 thru MTC7, and MTC10 thru MTC17. Only device names for MTC0 and MTC10 are actually needed. The rest I delete.

IPC Files in :PER

Because the peripheral directory is easily referenced (using the "@"), it makes sense to place IPC Files there. EXEC of course has it's IPC files there as does CEO.

On my Desktop I designed a CEO Secretary process to simplify remote mail checking. To keep in tune with my desire for simplicity, the Server Process creates its IPC file in :PER. The **Become Infos** privilege allows this to occur. (See Chapter 5) To "talk" to the CEO Secretary, I need just issue the CLI command:
Control @Secretary Mail?

It couldn't be any easier.......

Disk Tracing

To wrap up our discussions on the Disk world, I want to discuss disk tracing and then actually trace a disk (actually a diskette, because it is smaller). In order to trace a disk, we need to first review some of the directory structures covered earlier in this chapter and then explore these items in greater detail.

In order properly trace a disk, we need to first understand the relationship between the 5 **File Data Elements** (**FDE**'s). This is illustrated in Diagram 4-6.

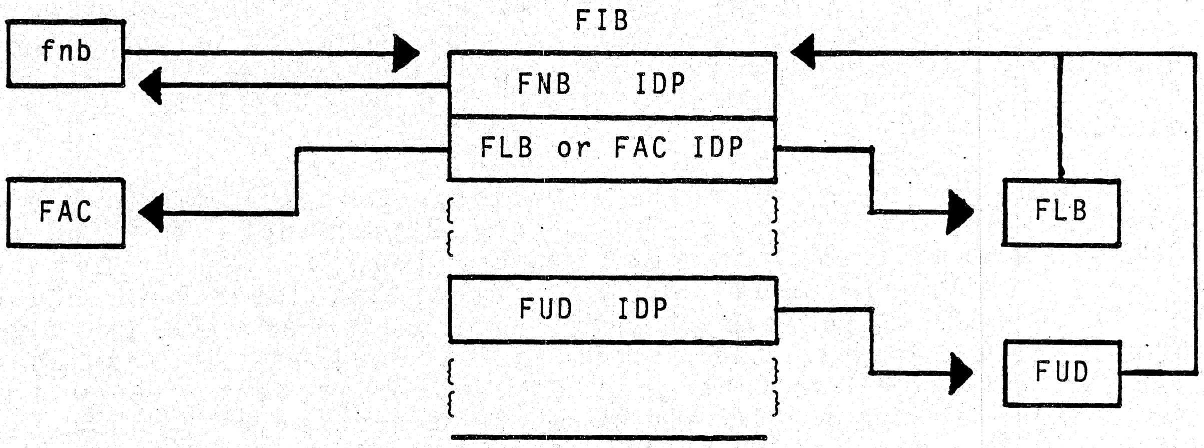
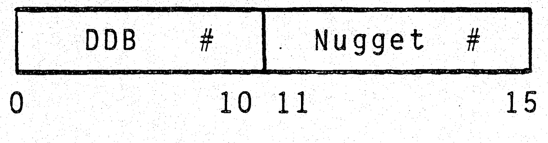


Diagram 4-6
File Index Block Linkage

As you can see in the above diagram, the **File Information Block** (**FIB**) is really the central control block for the file. It is located by following the link from the **File Name Block** (**FNB**). Once located, the FIB is used to keep track of a file's reference and status information.

As a side note, issuing the CLI F/S command simply lists the FNB entries for a given directory; F/AS forces an additional disk access to pick up the FIB for the "assorted" information.

All DDE's are linked via an Intra-Directory-Pointer (**IDP**). The IDP is broken into two segments: 1) Relative DDB# within the directory; 2) Nugget # within the DDB.



The DDE's Defined

"insivible directory space (DDB's) is utilized by the 5 kinds of DDE's described earlier. Alothough the various tYPES OF DDE's differ significantly, the first two offsets are always the same:

- Offset 0 - DDE Type (L. Byte) and Word Lenght (R. Byte)
- Offset 1 - IDP to FIB

There is actually a 6th DDE: the **FUI** - or **F**ile **U**nique **ID** - for LDU's It functions just like an FNB so is not given any special attention here.

FNB's

An FNB is a Type 1 DDE used to hold a filename. It is made up of one or more nuggets (3 max. per filename). In the initial nugget, we can hold up to 12 characters. In order to conserve disk space, filenames should idelly be 12 characters or less.

Type=1	Length
IDP to FIB	
"F	"I
"L	"E
"N	"A
"M	"E
<0>	

FAC's

An FAC is used to hold the acced control information for a file. It is made up of one or more nuggets. If the ACL is +... only, no FAC is assigned. In this case the ACL byte is in bits 11 - 15 of the FIB Status word.

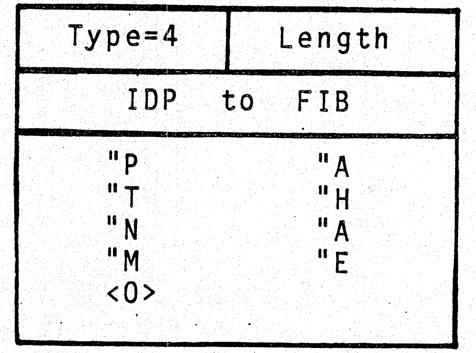
Type=2	Length
IDP to FIB	
"U	"S
"E	"R
"1	<0>
<ACL>	"U
"S	"E
"R	"2
<0>	<ACL>

ACL BYTE

	O	W	A	R	E
Bit	3	4	5	6	7

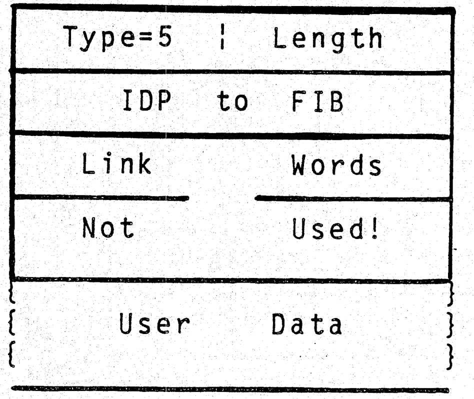
FLB's

The **FLB** exists only for files. The Bulk of its nugget space is used to point to the resolution filename. Because it is a link file, there is no FAC, the FLB takes its place.



FUD's

When a File User Data Area is created, it resides in the FUD occupying 17 nuggets. Although the UDA is only 128 words, the FUD is 136 words in length. Once created (via ?CRUDA), the UDA can not be deleted. It can only be accesssed via ?WRUDA and ?RDUDA system calls.



The Disk Information Block (DIB)

In order to trace a disk, we need an entry point for that disk. This gate is known as therun the filename through the hashing algorithm to locate the FIB for that directory. That in turn will point us to the DDB's for the directory. Once found, we can again run ted in the "Root" Directory DDB's are the FNB's for the files and the first-level of directories on the LDU.

To locate the files within a directory, we need to run the filename through the hashing algorithm to locate the FIB for that directory. That in turn will point us to the DDB's for the directory. Once found, we can again run the hashing algorithm to locate the FNB for the 2nd level of directory. We can continue this, ad-finitum, until we bottom out in the directory structure.

The DIB is not exactly a file per se', as it not pointed at by a Filename Block (FNB), although it does point to an FNB (the LDU name). In the latter portion of the Disk Information Block you can see the pointers to key system areas, including: The Bitmap, The SYSBOOT, The System Overlay Area, and the Installed System.

Rounding out the DIB is what is known as the "funny FIB" (it doesn't seem so hilarious to me). The difference between the "Funny" FIB and a regular FIB is that the "Funny" FIB does not contain the first 6 FIB words.

Finding the "ROOT" DDB's

The most difficult aspect of tracing a disk is in finding the "Root" Directory DDB's. Once this pointer is found, disk tracing is a repetitive snap, until you cross LDU boundaries (in the case of grafted LDU's); then you need to start over.

The Disktrace facility of the Watcher utilities not only allows us to conveniently display the Disk Information Block, but is designed to run the hashing algorithm, compute the IDP's and examine the the DDB's/DDE's, as well as a file's data blocks. First off, we need to display the disk's DIB with the Disktrace Command: **Block 3.**

At Offset 17 - 6 (ie. 11) of the "Funny" FIB (remember, it is 6 words less than a "standard" FIB), we find the 2-Word value FIFAH/FIFAL - the First Logical Address (all addresses are logical except for the "invisible space" at the "front" of the disk). This double-word is fixed at offset 0061/0062, so it is easily found. This address points to the first RIB of the "Root" directory for that disk. Now, all we need to do is run the hashing algorithm on the desired file to locate its DDB and we are in business. Diagram 4-7a (on the next page) highlights the key-words of the DIB for our purposes.

```
Physical Block: 000003
000   000003 050115 000000 030462 032063 000000 000001 000001 ...PM..1243.....
010   000002 000011 000050 000010 000000 001310 000000 000002 ......(.........
020   000000 000000 000000 000000 000000 000000 000000 000000 ................
030   000000 000000 000000 000000 000176 000000 000177 000000 ..........~.....
040   000412 000000 000000 000000 000000 000000 000000 000000 ................
050   000000 000013 000007 000000 000000 000000 011000 000000 ................
060   000001 000000 000200 000403 000001 000000 000022 000000 ................
070   001107 000000 000000 000000 000000 000000 000000 000000 .G..............
100   000000 000000 000000 000000 000000 000000 000000 000000 ................
110   000000 000000 000000 000001 000000 000000 000000 000000 ................
120   000000 000000 000000 000000 000000 000000 000000 000000 ................
*****
370   000000 000000 000000 000000 000000 000000 000000 000000 ................
```

Diagram 4-7a
The Disk Information Block (DIB)

Locating the Directory RIB

To find the directory RIB, we need to first change the block display mode to Logical and display the first logical block. This is done with the following Disktrace Commands:

```
Mode     Logical
Block    200
```

```
Logical Block: 000200
000   000000 000000 000000 000000 000000 000000 000000 000202 ................
010   000000 000000 000000 000000 000000 000000 000000 000204 ................
020   000000 000203 000000 000000 000000 000000 000000 000000 ................
030   000000 000000 000000 000000 000000 000000 000000 000000 ................
*****
370   000000 000000 000000 000000 000000 000000 000000 000000 ................
```

Diagram 4-7b
The "ROOT" Directory RIB

Filename Hashing and FNB Locating

Having found the directory RIB (Logical Block 200), now we need to "hash" the filename to determine the RIB offset the the DDB. For simplicity, let's look for the directory called "UTIL". Using Disktrace, we enter the Command: **Hash UTIL/7** (I chose a H.F.S. of 7 based on DIB offset 52).

Disktrace responds:

Hash Value is: 000003
Double the Hash Value (ie. RIB Displacement)**: 000006**

Now lets display that block with Disktrace Command:

Block 202

Diagram 4-7c gives us the results. At offset 0011 (underlined) is the IDP the FIB for the directory. Once we find the FIB we can locate the DDB's for UTIL.

```
Logical Block: 000202
000  000000 000000 000000 000000 000000 000000 000000 000000 ................
010  000410 000345 052524 044514 000000 000000 000000 000000 ....UTIL........
020  000000 000000 000000 000000 000000 000000 000000 000000 ................
*****
370  000000 000000 000000 000000 000000 000000 000000 000000 ................
```

Diagram 4-7c
DDB containing the "UTIL" FNB

Locating the FIB

The FIB is located by running the IDP formula and finding the DDB containing the FIB. Again let's utilize the services of Disktrace with the Command:

IDP 345

Disktrace responds:

Relitive Block #: 000007
Rib Offset to Block: 000016
Data Element #: 000005
offset in Block to DDE: 000050

DDB # 7 in the RIB (Block 200) points us to Block 204.

```
Logical Block: 000204
000   000000 000000 000000 000000 000000 000000 000000 000000 ................
*****
050   000000 000000 000000 000000 000000 000000 000000 000000 ................
060   000007 000000 000000 000000 011000 000000 000001 000000 ................
070   000205 000403 000000 000000 015054 067002 015055 067002 .........-n..-n.
100   000000 000000 000000 000015 000000 005670 000000 000000 ................
110   001010 000345 047520 000037 022053 000017 000000 000000 ....OP..$+......
120   000000 000345 047520 000037 022053 000000 000000 000000 ....OP..........
130   000000 000000 000000 000000 000000 000000 000000 000000 ................
*****
370   000000 000000 000000 000000 000000 000000 000000 000000 ................
```

Diagram 4-7d
DDB containing the "UTIL" FIB

Locating the next-level Directory RIB

Now that we have found the FIB for the UTIL directory, offset 0017/0020 - First Logical Address - (underlined in Diagram 4-7d) will point us to the directory RIB shown in Diagram 4-7e. From here on out it becomes pretty academic.

```
Logical Block: 000205
000   000000 000000 000000 000214 000000 000220 000000 000206 ................
010   000000 000212 000000 000216 000000 000000 000000 000210 ................
020   000000 000207 000000 000000 000000 000000 000000 000000 ................
030   000000 000000 000000 000000 000000 000000 000000 000000 ................
*****
370   000000 000000 000000 000000 000000 000000 000000 000000 ................
```

Diagram 4-7e
The "UTIL" Directory RIB

Locating "FILE1" in UTIL

Now that we've found the DDB's for UTIL, let's locate a file called "FILE1". Using the HASH command of Disktrace we get:

Command: **HASH FILE1/7**

Deskwatcher response:

Hash Value is: 000001
Double the Hash Value (ie. RIB Displacement)**: 000002**

DDB #1 in the RIB of course points us to Block 214.

```
Logical Block: 000214
000   000000 000000 000000 000000 000000 000000 000000 000000 ................
010   000410 000354 043111 046105 030400 000000 000000 000000 ....FILE1.......
020   000000 000000 000000 000000 000000 000000 000000 000000 ................
*****
370   000000 000000 000000 000000 000000 000000 000000 000000 ................
```

Diagram 4-7f
DDB containing the "FILE1" FNB

Finding the FIB for "FILE1"

Having located th FNB, let's take the IDP (underlined in Diagram 4-7f) and run it through the Disktrace IDP formula with the Command:

IDP 354

Deskwatcher responds:

Relative Block #: 000007
Rib Offset to Block: 000016
Data Element #: 000014
Offset in Block to DDE: 000140

DDB # 7 in the RIB (Block 205) points us to Block 210. At Offset 0140, we will of course find the FIB for FILE1.

At Offset 017/020 in the FIB (0157/0160) we will find the first logical block address of the file. Because there are no RIB's in this file (FIB Offset FIIDX (0021) - Left Byte is 0), we know that this block is the data Block. If there were RIB(s), this Byte would have a 1,2 or 3 as its value.

Displaying the Data Block (215) shows us that we have in fact found a macro file. Closer inspection will disclose the fact that I simply copied one of the Deskwatcher macro files as FILE1.

```
Logical Block: 000210
000  000000 000000 000000 000000 000000 000000 000000 000000  ................
010  001440 000141 000346 000000 015055 066603 000000 001104  ...a.....-m....D
020  000000 000000 000000 000000 000405 000000 000001 000000  ................
030  000211 000003 000000 000000 015055 066773 015055 066773  .........-m..-m.
040  000000 000000 000000 000000 000000 000000 000000 000000  ................
050  000000 000366 022122 047516 000037 000000 000000 000000  ....$RON........
060  001010 000341 022122 047516 000037 000000 000022 000000  ....$RON........
070  001440 000201 000353 000000 015055 066614 000000 001104  .........-m....D
100  000000 000000 000000 000000 000535 000000 000001 000000  .........]......
110  000213 000003 000000 000000 015055 066775 015055 066775  .........-m..-m.
120  000000 000000 000000 000000 000000 000000 000000 000000  ................
130  001010 000347 022122 047516 000037 000000 000000 000000  ....$RON........
140  001440 000041 000360 000000 015055 066561 000000 001104  ...!.....-mq...D
150  000000 000000 000000 000000 000377 000000 000001 000000  ................
160  000215 000003 000000 000000 015055 066776 015055 066776  .........-m..-m.
170  000000 000000 000000 000000 000000 000000 000000 000000  ................
*****    --- (Contents unimportant for this Example) ---
370  000000 000000 000000 000000 000000 000000 000000 000000  ................
```

Diagram 4-7g
DDB containing the "FILE1" FIB

```
Logical Block: 000215
000  041557 066555 062556 072012 041557 066555 062556 072040  Comment.Comment.
010  022461 022400 064563 020164 064145 020122 062546 071145  %1%.is.the.Refre
020  071550 020104 062554 060571 020126 060554 072545 005103  sh.Delay.Value.C
030  067555 066545 067164 020057 005151 062075 067040 060554  omment./Pid=n.al
040  066167 073563 020141 020120 064544 020164 067440 061145  lows.a.Pid.to.be
050  020163 062554 062543 072145 062012 041557 066555 062556  .selected.Commen
060  072012 050165 071550 005120 071157 066560 072040 050157  t.Push.Prompt.Po
070  070012 051545 060440 035120 062562 026133 020523 062541  p.Sea.:Per'[!Sea
100  056412 050162 067543 027504 062546 027511 067543 027502  ].Proc/Def/Ioc/B
110  066057 050162 064557 071075 030460 027516 060555 062475  l/Prior=10/Name=
120  053541 072143 064145 071057 051145 071551 062145 067164  Watcher/Resident
130  020072 042145 071553 073541 072143 064145 071072 042145  .:Deskwatcher:De
140  071553 057567 060564 061550 062562 027520 071151 067146  sk_watcher/Prinf
150  067457 022460 027445 027501 072564 067537 041046 005133  o/%0/%/Auto B&.[
160  020516 062561 026045 030445 026135 027504 062554 060571  !Neq'%1%']/Delay
170  036445 030445 055441 042556 062135 005120 067560 005000  =%1%[!End].Pop..
200  000000 000000 000000 000000 000000 000000 000000 000000  ................
*****
370  000000 000000 000000 000000 000000 000000 000000 000000  ................
```

Diagram 4-7h
First Data Block for FILE1

Disktracing Summary Points

Disk tracing is a powerful method for monitoring the AOS disk structure internally to see what is really going on. With Disktrace, virtually nothing on the disk is left "invisible".

Using the examples and text from this chapter, as well as the Control Block breakdown in Appendix B, you should be able to get the "scoop" on disk efficiency of your AOS system. If you want a more detailed look, an AOS Internals course is available from Innovative Data Systems. Or, bring me "in-house" for a day an we can "trace" your system together. As with any skill, disk tracing requires patience and practice. Go for it.

Chapter 5
The Preditor and System Performance

The Preditor and User Privileges

Although only partially related to System Performance, granting process privileges carefully can insure that no user dominates the system or has access to resources that could compromise system integrity. An example of privileges that could potentially comprise the system are of course Superprocess and Superuser.

At this point, I will assume that you have read the chapter pertaining to the Preditor utility and process privileges in the "How to Generate and Run....." reference manual, and will proceed here based on that assumption.

As a general rule, we can break the privileges down into the following three groups:

Group 1 - Privileges that affect the initial logon and simplify/enhance environment setup for the user.

Group 2 - Privileges that allow the user to indirectly impact the system environment and often System Performance.

Group 3 - Privileges that allow the user to directly impact system performance and operation.

Let's look at these in the order of importance to the system.

--- Group 3 Privileges ---

The list of privileges that offer direct system impact and are thereby labeled "dangerous" are:

- Superprocess
- Superuser
- Change Username
- Access Devices
- Use IPC

- Superocess and Superuser -

Superprocess and Superuser are obviously "dangerous" privileges to be handing out and yet in visits to many AOS shops, I find this privilege granted freely. In most cases, these privileges are hardly warranted.

As a case in point, the need for Superuser can be nearly eliminated by setting up **A**ccess **C**ontrol **L**ists thoroughly and accurately. Granted ACL's are expensive in terms of disk space, but if security is an issue , ACL's are the only way to go.

I remember working with a small group some years ago where the issue of "shall we, or shall we not grant all staff members Superuser" was debated heatedly for weeks. In this particular group however, there was an emotional issue at stake which overrode concerns for system security and integrity. Because this was predominantly an R&D group, those demanding unconditional "Super" privileges won out.

If system integrity is ever an issue, I vote in favor of extensive Access Control Lists, giving Superuser privilege to no one except a password protected (System-manager-type) Username to be used only in the event of an "emergency". The investment of the time and effort required to set this up is well worth your while. You won't fully appreciate it until the first delete command with a "+" or "#" template is averted.

A similar condition exists with regard to the Superprocess privilege. With this privilege you can terminate any process on the system, including PID 2: OP-CLI, which is of course indirect suicide. In addition to termination, you can block/unblock any process as well as change the priority and process type. These kinds of things might be necessary on a system-wide basis for a system manager and occasionally a system programmer, but hardly for the typical user.

The critical system processes are: PMGR, OP-CLI, EXEC and INFOS (not to mention the CEO Server Processes in the event you run under CEO). Changes in the parameters of these processes by unknowledgable users could severely impact system performance as well as system integrity.

Like Superuser, this privilege should be given **only** to those users that have an actual need to alter critical system runtime parameters. To qualify as such a user, you should be fully adept in the use of the knowledge in this manual.

Now, let's take a look at the less-critical, but nevertheless "dangerous" privileges.

- Change Username -

This privilege allows you to Proc a son process with a username of your choice. With this privilege available, you can get around not having Superuser by proc'ing yourself up with a username of a user having the ACL access, and of course this could be very undesirable.

A positive aspect of this privilege is in use on my Desktop system. I have written a macro called **Become** that allows me to proc a son CLI of another username in order to handle system problems and alter performance parameters (Ex: Invoking the System Tuning Screen of **Aoswatcher** requires the OP username).

For example, I can Become OP and issue Control @Exec commands to restart failed print jobs, DIR into "OP Access Only" directories and control batch streams to improve system thruput of my software development work. Without this privilege I would have to walk quite a distance to get to the OP console and "unlock" it. I can also "become" a user that is experiencing difficulty (while retaining my process privileges) to assist that user in fixing the problem.

- Access Devices and Use IPC -

These privileges allow what I call "hacker access" because with these capabilities, a competent programmer/"hacker" could get into the system and literally gain entire access to all system resources, including the internals of the AOS operating system itself.

Armed with IPC privilege and intimate knowledge of PMGR internals (taught in DG education courses), one could gain access to PMGR's control ports and either alter terminal operational parameters significantly, or "crash" PMGR altogether (and thereby the system) if bad control parameters are passed.

In the case of Access Devices, a good programmer with knowledge of MMPU I/O instructions and AOS internals could read/write any word of memory, grant illegal privileges, impact the operating system's performance or (more likely) force AOS into its Panic routines.

So, it should be rather evident that these are all dangerous privileges and should be restricted only to those that really have a need - and this is pretty rare.

--- Group 2 Privileges ---

These privileges can indirectly impact system performance, although normally they cannot compromise system integrity; and, there are a few exceptions to this rule. The group 2 list is rather extensive.

The list of privileges that make up Group 2 are:

- Change Priority
- Change Type
- Unlimited Sons
- Number of Sons
- Create without Block
- Memory [size]
- Priority
- Disk Quota
- Use Console
- Use Batch
- Use Modem
- Use Virtual Console
- Access local Devices Remotely
- Become Infos
- Max Qpriority

Use Console, Use Batch, Use Modem, Use Virtual Console and **Access Local Devices Remotely** are more or less convenience privileges, although they can generate system impact.

For example, the use of batch privileges gives the user the opportunity to queue up more processes after the max. son limit has been reached, thereby averting that restriction to some degree.

Console access can impact a predominantly batch environment by allowing unauthorized users access to the few consoles that may be available. In the case of virtual console access, remote logon becomes available opening the door to "hacker" access. The same goes for accessing devices remotely.

The remainder of Group 2 privileges allow (to some degree) system performance to be impacted. Let's look at these privileges more closely.

- Change Type -

The Change Type privilege allows a process to make itself Preemptible or Resident, thereby enabling higher Memory and/or Process priority. This can force AOS into Memory Contention, thereby impairing System Performance. As a reminder, the default process type is swappable, allowing the AOS Scheduler and Core Manager to control System Performance based upon process behavior. If too many resident processes are proc'ed, a memory "deadlock" can occur, thereby "crashing" the system.

- **Memory** [size] -

Along with "Change Type", the Memory [size] parameter can affect Memory Management Performance. Rarely have I seen this parameter set to anything other than 32 and this could be a mistake. Recognize that some software is written to extend its unshared memory space (with the ?MEMI system call) to the maximum available enabling it to run faster; at the expense of other processes on the system (and of course eventually at its own expense). Therefore, it is wise to set this figure no higher than absolutely necessary. Of course, when in doubt as to what program(s) the user(s) will run, set the memory to 32 pages allowing the largest possible program size.

- Priority & Change Priority -

These privilege declare the initial process priority at logon and then declare whether or not a process can raise its priority higher than its initial [assigned] priority. A process can always lower its priority, but I have never seen that happen!

You know the old law: give a process an inch and it will take a mile? Well, stated another way: given the choice, nearly all users will run at Priority 1. If they are Resident or Pre-emptible as well, then they will be competing with such processes as PMGR and this could cause REAL problems.

Remember, Priority Structure is one of the ways you can control System Performance . Make use of this privilege wisely.

- Number of Sons -

This privileges puts a ceiling on the number of son processes a given user can create. Obviously, the more sons on the system the poorer the performance will be. As a general rule, unlimited sons are rarely needed in the typical user environment.

As I write this manual I am also Beta-testing the **Deskwatcher** System Performance analyzer. With the unlimited sons privilege, I am able to proc 25 - 40 dummy processes to measure system load and AOS performance. Normally however, 3 sons is a good limit. With 3 sons available, I can be editing a source file under SED (Son #1), break out to the CLI (Son #2), and bring up the current version of **Deskwatcher** (Son #3) in order to test one of its features. When I have completed my test, I can terminate **Deskwatcher**, and return back to my editing session to make the source-code changes needed to enhance the software.

A good rule of thumb is to keep the son limit down to a workable minimum; especially if you run software that tends to proc son processes on the fly. However, if you institute too strict a policy on son processes, you will have more user complaints than you care to handle. There is almost nothing worse than incurring the wrath of the user base; especially in a Desktop Environment.

- Create Without Block -

On many systems, memory is not a problem, but active process are. If this is your scene, then create without block is a quick and dirty solution. Without this privilege a user can then only have one active process. The memory will still be impacted by the existence of this process, but at least the process scheduler will be given a break.

- Become Infos -

This privilege is misleading in that it simply grants the user process the privilege to create IPC files in the :PER directory, and that's all. The value of this privilege is that it can simplify IPC file look-up for other processes and has the disadvantage of increasing the directory search overhead of :PER. (Discussed in Chapter 4) Because this directory is frequently accessed by the system, performance will suffer if too many IPC files reside in :PER

- Disk Quota -

Because this parameter controls the UDD cpd maximum size, it can indirectly affect System Performance. Only if the user attempts to fill up the allocated CPD space will a large disk quota be a problem. Remember, this value simply sets the CPD maximum. It does not actually allocate the disk blocks.

The most important consideration here is the overall disk space utilized, not the CPD size limits. It is worth reiterating however that if the allocated disk space exceeds 80% of the available disk blocks, then System Performance will begin to suffer drastically.

- Max Qpriority -

This figure limits the max Qpriority on batch jobs and is therefore of use to the system manager in controlling batch queue utilization. Proper use of the Qpriority setting can favor batch processing for certain users, enhancing their productivity.

--- Group 1 Privileges ---

Most of the Group 1 "privileges" are not privileges per se', but instead are environment directives describing the initial operating environment of the user or are of documentation use to the system manager.

The Five Group 1 directives are:

Username	Initial IPC File
Change Password	Program
User Comment	

- Initial IPC File -

When a program is invoked at logon, a file can be passed to it in order to get it started; this is called the IPC (or Initial Process Communication) file. A restriction on this file is that it can be only one block in length, but this causes no hardship to astute programmers and System Managers.

If the initial program is the CLI (as it usually is), then this file is usually a "logon macro" used to get the user environment setup and give such information as the "local system news" and possibly describe current system activity. Under CLI, this file is often not big enough to handle all the logon requirements of a given user. The way around this is to have the initial logon macro simply be an invocation of a second macro which can be of any length.

In my case logon occurs via a set of 4 macros totaling approximately 3800 bytes and is invoked by initial IPC file :Staff:$Ron:Logon.Cli which has one command in it: $Logon; this being the first of 4 macros: $Logon, $Logon1, $Logon2, $Logon3. In most installations I've visited, the initial IPC file is rarely used to its full potential.

- Username -

On a properly "designed" system, choice of Usernames is no trivial matter. Remember, Access Control Lists are enforced based upon user name. That makes selection of Usernames a potentially critical item if system security is an issue.

For example, on my desktop machine, any username preceded by a "$" denotes System Programmer or System Manager status. A username followed by a ".Staff" indicates a corporate staff member, giving additional access to privileged mail and CEO. A username followed by ".Vip" indicates preferential status, but no "super" privileges and access to staff mail or CEO. All other "low-life's" simply use their name to logon.

- Program -

Normally, when users log onto the system they are brought up under the Command Line Interpreter (CLI), but this is not always a desirable case. To begin with, under CLI a user can do things that would impact the system and/or go into directories that might potentially put the system at risk. Another possibility is that the CLI is actually too complicated for the user; a case not all that unheard of. Therefore, you can actually bring a user up under almost any program.

Some cases of not bringing up the user under CLI might be in the case of a predominantly CEO shop or a shop that runs a reservation system most of the time. In these cases you simply bring the user up directly under this software and when they terminate the program they will be logged off. In one educational institution I spent some time at, beginning programmers were brought up under Extended Basic when they logged on to the system and those desiring access to the "games directory" logged on with a username of "Games.Vip" and were brought up under a menu-driven program designed to allow them to select the game they wished to play.

- Change Password -

The change password privilege is useful for keeping the computer game interesting. I will at times change my password when my mood changes to add spice to life. I also change my CLI prompt from time to time for the same reason. Never is there a dull moment at **Innovative Data Systems**.

--- Preditor Summary ---

To sum up this discussion on the profile and privileges, I want to remind you that assigning process privileges IS an area that you have <u>direct</u> control over. Other areas concerning system performance may require specialized knowledge, but controlling process privileges is something most anyone can do at almost anytime. It is up to you whether you avail yourself of this capability or not.

```
************************************************
*                                              *
*                                              *
*                 Chapter   6                  *
*Hardware Configuration and System Performance*
*                                              *
*                                              *
************************************************
```

Improving System Performance

Thus far in this book we have covered a number of different aspects of AOS system performance, although in many cases things haven't been exactly spelled out. The purpose of this chapter is to offer some additional insight and recap some of what has been stated earlier. Bear with me if I seem to be repeating myself.

Performance improvement can be broken down into three major segments which although different, are integrally related:

- Hardware Configuration and Aosgen parameters
- Operational Parameters
- Operational Procedures

Configuration and AOSGEN

Configuration and Aosgen often go hand-in-hand and can make a significant difference in the performance of your AOS System. Configuration is simply selecting the proper devices and hardware options for your system. Normally, configuration precedes the Aosgen stage, but not always.

Aosgen is the step wherein you tailor the operating system to fit the configuration of your hardware. Normally this follows the configuration step. You can of course choose to purposely "over-gen'" your system in anticipation of devices to soon be added. Doing this eliminates the need for re-gen'ing in the future when these devices are actually added. The drawback to this is of course that extra memory will be needed by the operating system; even though it is not being used.

Operational Parameters

Operational parameters are separate from configuration and Aosgen in that they are pretty much independent of the two. Operational parameters include such things as ACL's, file element size, searchlist changes and the like.

Operational Procedures

Operational procedures can often make a noticeable difference in the performance of the system. Such procedures include: Disk Compression, "Job" Scheduling and System Process Load.

Performance Tips

Below is a list of things you can do to improve system performance at the Configuration, Aosgen and Operational parameters level. The following pages detail what you can do in each of those areas to enhance performance.

- Add additional Memory
- On older machines, be sure the memory is interleaved
- Increase the cache buffer count
- Patch the OTMIN location to keep AOS overlays resident for longer periods of time
- Add additional Disks and/or Controllers
- Use a BMC when you have more than one disk controller
- Keep searchlists small and identical
- Use + template on Acl's or eliminate Acl's altogether
- Keep directory depths minimal
- Optimize hash frame sizes on directories
- Keep filenames under 12 characters
- Choose large file element-sizes to reduce file-indexing
- Make continuously used files contiguous
- Place Bitmap and Overlay area in the middle of the LDU used area to reduce the seek distance for that drive
- Make I/O buffer sizes as large as possible to reduce I/O requests
- Occasionally compress your disks
- Install a DCU when using more than 8 terminals
- Cut the baud rate to 4800 on non-DCU machines with many console lines
- Gen the RTC at the lowest frequency possible
- Utilize batch processes for compute-bound jobs; non-interactive processes significantly hog cpu time. Using batch forces these jobs to single-thread.

Additional Memory

Adding additional memory boards/cards will most always improve system performance. In years past, this was not always an economical move. Now days with declining memory prices, adding physical memory is a very practical thing to do.

Interleaving Memory

On old AOS machines, the memory boards were not interleaved. This would tend to slow down the memory reference cycles. Then, an innovative announcement from Data General was that memory boards could be 4-way interleaved, speeding memory access by about 4X. The newer Eclipse CPU's are already interleaved and need no such change. If you are uncertain whether or not your memory is interleaved, contact your field engineer.

Terminal Considerations

Although many things contribute directly to system performance, terminal operation is in a way the most "visible" to users (no pun intended). If terminals are operating inefficiently, PMGR will indirectly suffer. Being that PMGR is a resident priority 1 process, should it run inefficiently, it could indirectly impact process performance on the rest of the system.

Three things should be given consideration when setting up terminals on an AOS system:

- Terminal Characteristics
- Baud Rate
- Use of a DCU device

Terminal Characteristics

Terminal characteristics can make or break a system from the psychological standpoint of users. It is important to read the reference manual(s) for your terminal(s), as well as the AOSGEN section of the "How to Load and Generate" manual to be sure that everything is in sync.

Some years ago, I was part of a research project to determine what people like/disliked with regards to using a terminal. Unfortunately, it is a highly subjective topic, often dependent upon the application software driving the terminals themselves. However, it is best to be alert to people's reaction to: beeping, highlighted fields, field placement, redundancy of data on the screens (or lack there of), how **much** data is one each screen and what the baud rate is. Maybe someday, I will write a whole book on this topic alone.

Baud Rate

Baud rate can have quite an impact on system performance; especially on a system with more than 8 terminals. There is a tendency to run the baud rate as high as possible. Although screen I/O speed is seemingly enhanced, the system is easily bogged down trying to supply large amounts of data to several terminals in a timely fashion. Remember: the CPU can only process data so fast.

It is important to ask yourself whether or not the users really need to run at 9600 baud. Quite often 4800 baud is more than sufficient. On screens with "small" menus or screens where just a few fields are plugged in, 2400 baud may even suffice.

It is worthwhile looking at the application software (if you have the source code available) for different ways to enhance the screen I/O. For example, it is common practice to access all the needed data and THEN toss out the screen. A more effective approach would be to access some of the data, put it on the screen. While the user is studying the first part of the screen data, the software can be accessing the second (and subsequent) pieces of data; updating the screen. Overall, the amount of CPU time will increase some with this method, but the thruput will improve as a result. The additional processing steps will be done while other terminal screens are receiving their data.

The above method DOES require planning and possibly some investigation into psychological effects of "overlayed" screens, but it is well worth the investment. After all, what good is a system if the user's refuse to utilize it?

Adding a DCU Processor or an IOP

On non-Desktop systems a DCU process can be added to enhance terminal performance. There are two types of DCU facilities: the DCU-50 and the DCU-200. The DCU-50 speeds terminal I/O in general and the DCU-200 places most of the PMGR functionality into the DCU processor off-loading the central processor - the resident PMGR (called the Wart) is invoked ONLY when the DCU is incapable of handling a required function; and this is rather seldom.

The M/600 system comes equipped with an IOP (or I/O Processor) and is an option on the C/350 processor. Essentially, the IOP is a super-intelligent DCU, running in conjunction with IOPMGR as the PID 1 process. Its purpose is to enhance the interrupt handling of slow-speed I/o devices (eg. consoles, printers).

The Real-time Clock (RTC)

AOS runs best when the Real time Clock is gen'ed at the lowest frequency possibly. Higher clock frequencies generate more interrupts per second slowing the system down. For example, a 10 hertz clock generates 10 interrupts per second; a 100 hertz clock generates 100 interrupts per second; a 1,000 hertz clock generates a 1,000 interrupts per second. Unless your software really needs 100 or 1,000 hertz timing accuracy, it is best to choose a 10 hertz clock speed.

Cache Buffer Count

Adding extra cache buffers via AOSGEN can make quite a difference in system performance. Use of the word "cache" here is in a way a mis-nomer. The word Cache usually refers to faster memory, which these buffers aren't. AOS cache buffers get their name simply because they are core-resident. To understand why cache buffers make a difference, I researched AOS's block I/O algorithm scheme and found the answer. When a block request is made of AOS the following checks are made:

- The block has just been read into a System Buffer
- The block has been recently BLM'ed into a Cache Buffer
- The I/O for the block is currently in progress
- The block needs to be read (the request is queued)

The greater number of cache buffers available to the system, the greater chance of the block already being core-resident. Remember, one of the axioms of the AOS I/O world is that physical disk I/O is done **ONLY** as a last resort. Considering that the full 128 buffers require only 32kw of memory, it is well worth the investment of gen'ing in the maximum. If the loss of 32kw greatly impacts the memory management of your system, then you are probably overdue for extra memory to be added anyway. Using the Watcher's Memory Information screen will help point this out.

To demonstrate the cache buffer improvement, I added a new IOSTATS screen to Watcher utilities (now available in Release 3.0). This screen displays (amongst other things) the percent of logical vs. physical I/O requests and the percentage of I/O wait. Figure 6-1 demonstrates the effects of increasing the number of cache buffers. A visible manifestation of increasing the cache buffers is that program load time is significantly reduced. As you can see, using cache buffers is an inexpensive way to enhance disk I/O performance.

Before permanently changing the cache buffer count (via AOSGEN), first re-boot the system overridding default specs, specifying a larger number of cache buffers. When you find the right number for your system, then and only then re-gen the system.

Patching OTMIN

There is a memory location internal to AOS called OTMIN. This location specifies how long AOS will keep system overlays memory-resident. By default it is set to 74 (octal) - 5 seconds. If you have a sufficient amount of memory, it might be worthwhile to patch this to a much larger value; say 5 minutes. The following commands demonstrate how this is done:

```
X Dedit/s=Operating_system_name.ST  Operating_system_name.SY

+ OTMIN: 00074 + 454

+ Bye
```

This change is "officially" unsupported by Data General but has been known to work for over 6 years and seems to be "harmless" to the rest of the operating system. If memory is tight, this patch may trigger memory contention. If you re-gen the system, you will of course have to re-apply the patch. Needless to say, make a backup of your .SY file be-4 applying the patch.

Additional Disks and Controllers

Additional disks and controllers will most always improve system performance. This topic is well covered in Chapter 4 so I will not repeat myself here. Desktop considerations will be discussed at the end of this chapter.

Searchlists

Although they are rather useful, searchlists can be quite expensive in terms of system performance. The longer the searchlist, the more disk accesses that will be required in the event of a non-existent file, only to come up short. A few points are worth noting:

- Keep no more than 3 directories on your searchlist
- Do not put :PER on the searchlist; instead reference that directory with the "@" prefix
- Keep the same directories on your searchlist as other users. That reduces the AOS internal memory space required to hold the searchlist.

Directory Structures

Directories allow you to logically subdivide your disks so that files can be more easily found. Most systems have poor organization of their disks. Directories are not used enough. Yet on the other hand, other systems are over-directoried. Directories do have their place, however they also need to be controlled properly.

While many directories are useful, it is wise to keep the directory **depths** as shallow as possible. Nested directories result in lengthy pathnames to get to inferior directories. This inevitably slows the system down because the superior directory structures must first be scanned. This scanning requires time to read the directory blocks and a place to hold this information (AOS memory).

Choosing a proper Hash frame size (as we saw in chapter 4) will of course lend itself to more efficient directory structures. Likewise, making directories into CPD's allow you to monitor how much space is being utilized by a given set of files.

Filenames and File-basics

With files, there are a number of things worth considering. The information in this section has been discussed in chapter 4 but is worth summarizing here. The things to be conscious of are:

- Keeping filenames under 12 characters will require only one file data element (FDE) worth of space in the directory structure.
- Choosing large file element-sizes will reduce the amount of indexing within that file structure. In fact, making the file contiguous (ie. element-size = file size) will eliminate <u>all</u> index levels for that file.

- Using + Acl's will eliminate the FAC DDE for that file. In fact, Aos-gen'ing out Acl's altogether (if they are not absolutely needed) will also improve performance.

Bitmap and Overlay Area

Placement of the Bitmap and [system] Overlay area can make a significant difference in AOS performance. Ideally, these structures should be placed in the middle of other commonly used files. This will tend to minimize head movement on those drives. By default DFMTR chooses a location about 3/8's of the way into the LDU.

I/O Buffer Sizes

Carefully choosing I/O buffer sizes can easily affect performance of the system, or at least the application of which those buffers are a part.

In general, the rule for I/O buffer sizes is to make them as big as possible. This will allow the individual application to run more efficiently. Needless to say, there is a tradeoff between large buffer sizes and overall system performance. Making buffers too large will improve the performance of the individual application, although often at the expense of overall system performance. This is something to be considered.

With the Watcher utilities, using Screen #5 (Disk Statistics) we can monitor the ratio of controller I/O requests to actual blocks read/written. If the ratio of reads/writes to requests is very high, then overall the buffering schemes are probably rather efficient. A low ratio indicates that small buffering is causing additional I/O requests. I/O requests of course not only involve the device time, but include AOS Ghost and internal processing as well. On an I/O bound machine, this becomes very noticeable rather quickly.

Increasing the application program buffer size(s) will most surely improve performance, up to a point. Another possibility is to multi-task the I/O routines in the program, having a separate task handle the I/O for each file. This gets a little tricky to write, but can really improve the I/O speed of a given application. Likewise, using Shared Page (SPAGE) I/O can dramatically improve an application's performance.

For large files, increasing the file element size will decrease the number of transparent RIB accesses and therefore reduce the number of read requests to the disk.

Disk Compression[1]

As time goes by, disk drives tend to become rather fragmented, the disk blocks being scattered all over the Ldu. Fragmented disks caused system performance to be impacted due to the increased seek time on the disk drive.

The solution to disk fragmentation is to compress the disk drives from time. Disk compression consists of dumping your disk(s) in directory and/or access order, reformatting the disk and reloading the data. This tends to reduce or eliminate the disk fragmentation, thereby reducing the seek time. Although time consuming, disk compression is well worth the investment.

The question of course arises as to when it is necessary to do a compression. The best place to go for answers is to the bitmap to the disk itself. If you are experiencing mild fragmentation, you will see a "ragged" bitmap. As disk fragmentation continues to get worse, the "holes" in the bitmap will be filled and the disk will deceptively look un-fragmented. Therefore, examine your bitmap from time to time and also keep an eye out for disk "chatter" and the system slowing down; two signs of disk fragmentation. With the Watcher utilities, you can not only examine the Bitmap, but you can produce a hard copy listing of it for your documentation needs.

On my Desktop, I typically compress the disk(s) every 3 months or so. Doing this usually improves the slipping system performance by about 25%. On a Desktop, that is significant and therefore well worth the time taken. The process can be automated with macros that run under batch while you are off doing work on other drives.

As a side point, disk compression can often have quite an impact on drive busy and controller interference statistics. As the seek time decreases, the drive and controller will be busy for shorter periods of time.

---

1) For more information on compression, consult my articles in the June, Sept. and Oct. 1986 issues of data base monthly magazine.

Desktop Configuration Points

We can't even begin to discuss System Performance without looking at the issue of proper hardware configuration. Spending vast amounts of time dealing with System Performance is worthless if the hardware is under-configured. So briefly, I want to approach this from a number of different angles.

Let's use as an example the system that this manual is being written on. This Desktop is a Model 10sp with: 1240kb, a 38mb Winchester drive, a 4 Line Usam, a tape cartridge drive for medium-volume backups a Cipher 100 1600 BPI reel tape running off of an ICI DMT1 tape controller, two 5 1/4" floppy diskettes for writing text backup, a logic expansion chassis (for future "secret" projects), a second console, a letter quality printer, a dot-matrix printer (soon to become a line printer) running on an ICI DLP1 buffered parallel interface card, and a Signalman 300/1200 Baud modem. It could be argued that we have overdone ourselves and yet I tend to disagree. Adding a second disk drive several months ago made a significant difference in the performance of the system.

First off in order to run at all, AOS requires at least 1 hard disk, and a second drive is recommended. The initial desktop systems were configured with only 15 mb. disk drives and yet AOS installation alone requires approx. 13,500 disk blocks - close to 1/3 of a single disk. Even with two drives, space is very limited.

Luckily, since the initial release of the desktop, a 38mb. and 78mb drives have been added and 120mb. drives are reportedly now available as well. Unfortunately, hardware design of the desktop allows only one disk controller with a maximum of two drives on that controller. Therefore, it is recommended that you include the largest affordable drives to begin with; because you can't configure different drive types on the same controller.

In general, additional disk drives - if properly utilized - will significantly enhance system performance. This becomes even more significant in the desktop environment due to the limited drive capacity and the 2 bit data bus on the micro-I/o card.

On any system, disk seek time is always <u>the</u> problem. Therefore, two drives along with proper file placement can dramatically enhance system performance. In a single drive configuration, file placement becomes an important issue as was pointed out in Chapter 4.

Desktop Disk Drives

On the Desktop we are severely limited to our disk choices. Ignoring the diskette drives, we are limited to one disk controller supporting only 2 disks. In most respects, the Desktop systems resemble the S-10, S-20, S-30 series except that the Desktops have been endowed with a "crippled" disk controller and data-bus.

The winchester controller has been designed with 7 sector interleaving and small hardware buffering which slows the disk considerably. Additionally, the CPU has been designed with a 2-bit data-bus. Although this is probably adequate for character devices, it is disastrous for block devices. A 2-bit data-bus means that 8 memory fetches are required to bring a single word of data into memory. In the case of a disk block, 2,048 fetches are needed to complete the data transfer.

If you would like an in-depth understanding of the desktop disks, I recommend reading the Disk Technical reference manual. It is loaded with a host of "internal" information and therefore makes little sense to repeat it here. The manual is available from the DG TIPS department and is well worth the investment. What follows is a summary of the main points addressed in that manual, although this is hardly conclusive.

The Desktop Disk Controller Exposed

After just a few short pages into the disk technical reference manual I began to understand what the disk bottleneck is all about. Although I can get no verification from DG on this, the following explanation seems to be accurate.

The disk controller is sector buffered to allowed data transfers to/from the controller buffer and drive at 625kb/sec. Unfortunately, the Data channel speed is a mere 146kb for reading and 171kb for writing. The need for sector interleaving becomes very clear: the disk needs a "brake" mechanism so as not to overrun the data channel. This inept data channel speed seems to be a nasty side-effect of using a 2-bit data bus, instead of the usual 16 bits, or at least 8 bits. Figure 6-2 illustrates this. When compared to the Mv/4000-DC (3.0 mb./sec) and especially the little "Bulldog" Mv/2000-DC (8 mb./sec no less!), it becomes pretty clear that the desktop's disk performance was no accident.

For my system, while deciding upon the appropriate configuration for our Desktop, we opted to install a single 38 mb. drive over two 15 mb. units. Although there are advantages to using two disk drives instead of one, we needed the extra space, and the 38's are faster drives. Besides, if we needed to upgrade for more space, we would have to trade in our whole disk sub-system to eventually take advantage of the 38's speed.

The % of Busy and % of Interfered Requests give us an accurate idea of how much time AOS is "twiddling its thumbs" waiting to begin a disk request. This is overhead above and beyond the Disk I/O timings. The timing figures you read in the reference manuals assume an ideal scene wherein the program is staggering its I/O requests so precisely that it never makes a request while the drive or the controller is off handling a previous I/O transfer.

With a single disk sub-system, our Desktop was experiencing about a 30% Busy figure to the disk drive. This means nearly 1/3 of the time a request was made to the disk, there were one or more requests already on the I/O queue for that drive.

The disk controller can only process one data transfer at a time - serially - although each drive can seek independently. The other requests have to be placed on the AOS I/O wait queue. The Average-Q-length * Average-I/O-request-time (6.2 ms computed from the disk tech. reference) gives us an idea of how much time is being wasted waiting for the disk. Add to this: Average-Seek- Distance (in cylinders) * Cylinder-seek-time (per cylinder) and all of a sudden these "storybook" figures become unbelievably real and painfully slow. At least without a second drive on the system, there can be no interfered requests.

When we added a second drive, these figures changed significantly. To begin with, the % of Busy dropped to approximately 16% for drive DPN0 and was about 8% for drive DPN1, yet both drives were experiencing controller interference of about 5% of the time. Controller interference means that although the requested drive itself was not busy, the controller is tied up with a data transfer to the other drive. When we piled four users on the system, the % of Interfered Requests jumped to a whopping 40% - 45% for both disks. This was from word processing, assembling programs in a batch stream and running CEO, all concurrently.

Although the Avg. seek distance was now a mere 52 cylinders for Dpn0 (down from 94 cylinders), and 23 cylinders for DPN1, the system bottle-necked somewhat due to the controller interference. A faster controller (such as that on the MV/4000-DC even) would make a HUGE difference here.

Summary

Although the Desktop disk drives are indeed crippled, the desktop systems nevertheless represent some of the faster, more efficient desktops in the computer marketplace today. And, they are an excellent gateway into the world of AOS. They should not be overlooked.

```
***********************************************
*                                             *
*                                             *
*                 Appendicies                 *
*          Appendix A --- Appendix B          *
*                                             *
*                                             *
***********************************************
```

Reccommended Reading

- Advanced Operating System (AOS) Programmer's Manual
  by Data General - 093-000120

- How to Load and Generate Your AOS System
  by Data General - 093-000217

- Programmer's Reference Manual Eclipse-Line Computers
  by Data General - 014-000626

- 16-Bit Real-Time Eclipse Assembly Language Programming
  by Data General - 014-000688

- Advanced Operating System (AOS) Link/LFE User's Manual
  by Data General - 093-000254

- Advanced Operating System (AOS) Macro Assembler Reference Manual
  by Data General - 093-000192

- Advanced Operating System (AOS) CLI User's Manual
  by Data General - 093-000122

- Learning to Use your Advanced Operating System
  by Data General - 069-000018

- Model 10 and 10/SP Computer Systems Technical Reference
  by Data General - 014-000766

- Model 20 and 30 Computer Systems Technical Reference
  by Data General - 014-000767

- Model 6271 Disk Subsystem Technical Reference
  by Data General - 014-000768

- Cartridge Tape Subsystem Technical Reference
  by Data General - 014-000752

- I/O and Interfacing Technical Reference
  by Data General - 014-000774

- Using AOS on Desktop Generation Systems
  by Data General - 069-000058

- Operating Systems: A Pragmatic Approach
  by Harry Katzan Kr. --- Van Nostrand Reinhold Company

- Operating Systems: A Systematic View
  by William Davis --- Addison-Wesley Publishing Company

- Computer Performance Measurement and Evaluation Methods:
  Analysis and Applications
  by Liba Svobodova --- Elsevier Computer Science Library

Reccommended Reading

Chapter 1

A) Programmer's Reference Series - Eclipse Line Computers
The Generic "Blue Book": 014-000626-02 (1980)

- Appendix I
- Pp. 3-31 to 3-41

B) Model 10 and 10/Sp Computer Systems - Technical Reference
Manual 014-000766 [1y

- Pp. 1-41 to 1-61
- Pp. 4-6 to 4-7, 4-9, 4-20 to 4-27
- Chapter 5

C) Model 20 and 30 Computer Systems - Technical Reference
Manual 014-000767

- Pp. 1-29 to 1-46
- Pp. 4-3 to 4-5,4-11, 4-24 to 4-32
- Chapter 5

Chapter 4

A) Model 10 and 10/Sp Computer Systems Technical Reference
Manual 014-000766 [1]

- Pp. 2-28 to 2-60
- Pp. 4-10 to 4-12
- Pp. 4-14 to 4-16

B) Model 20 and 30 Computer Systems - Technical Reference
Manual 014-000767

- Pp. 4-33 to 4-38
- Chapter 7

1(Section 4 is very technical but is worth reading to pickup the data that is understandable)

File Information Block Layout

```
01      FINLP          ; Pointer to first FNB (IDP)
02      FIACL          ; Pointer to FAC (IDP)
02      FILBP          ; Pointer to FLB (Link Only - IDP)
03      FIUID          ; Unique ID
04      FITCH          ; File Creation Time (Hi)
05      FITCL          ; File Creation Time (Lo)
     ;
     ; FISTS TO FIIDR comprise the "Funny Fib"
     ;
06      FISTS          ; File Status
07      FITYP          ; File Type (R. Byte) and Format (L. Byte)
10      FICPS          ; File Control Parameters
10      FIHFS          ; Hash Frame Size (Directories)
10      FIDCU          ; Device Code (L. Byte) Unit # (R. Byte)
11      FIFW1          ; Future EOF Extension
12      FIFW2          ; Future EOF Extension
13      FIEFH          ; Last Logical Byte - for EOF (Hi)
14      FIEFL          ; Last Logical Byte - for EOF (Lo)
15      FIDFH          ; Data Element Size (Hi)
16      FIDFL          ; Data Element Size (Lo)
17      FIFAH          ; First Logical Address (Hi)
20      FIFAL          ; First Logical Address (Lo)
21      FIIDX          ; Current/Max Index Levels L/R
22      FIIDR          ; Count of inferior directories
23      FIFUD          ; Pointer to FUD
24      FITAH          ; Time Last Accessed (Hi)
25      FITAL          ; Time Last Accessed (Lo)
26      FITMH          ; Time Last Modified (Hi)
27      FITML          ; Time Last Modified (Lo)
30      FIFW3          ; FCB Address Extension
31      FIFCB          ; Virtual FCB Address or (0)
     ;
     ; --- For CPD Entries ---
     ;
32      FICSH          ; Current Size (Hi)
13      FICSL          ; Current Size (Lo)
34      FIMSH          ; Max Size (Hi)
```

Disk Information Block Layout

00	IBREV	; DIsk System and File System Rev. #
01	IBTYP	; Disk Unit Type
02	IBSTS	; Status Word (Per Unit Flags)
03	IBIDH	; LDU Unique ID (High)
04	IBIDM	; LDU Unique ID (Middle)
05	IBIDL	; LDU Unique ID (Low)
06	IBSNP	; Sequence # of the PU in the LDU
07	IBNPU	; Number of PU's in the LDU
10	IBNHD	; Number of Heads
11	IBNST	; Numbert of Sectors per Track
12	IBNCY	; Number of Cylinders
13	IBVIS	; Start Disk Addr. of Disk Space
14	IBNBH	; Number of Visible Disk Blocks (Hi)
15	IBNBL	; Number of Visible Disk Blocks (Lo)
16	IBBTH	; Phys. Addr of Bad Block Table (Hi)
17	IBBTL	; Phys. Addr of Bad Block Table (Lo)
20	IBBUI	; 10 Word Unique ID for N.C.
32	IBLDF	; LD Flags
33	IBNMH	; Disk Address of Name Block (Hi)
34	IBNML	; Disk Address of Name Block (Lo)
35	IBACH	; Disk Address of ACL Block (Hi)
36	IBACL	; Disk Address of ACL Block (Lo)
37	IBBAH	; Disk Address of Bitmap (Hi)
40	IBBAL	; Disk Address of Bitmap (Lo)
41	IBSBH	; System Bootstrap Address (Hi)
42	IBSBL	; System Bootstrap Address (Lo)
43	IBSSB	; Size of System Bootstrap Area
44	IBOAH	; Address of Overlay Area (Hi)
45	IBOAL	; Address of Overlay Area (Lo)
46	IBOAS	; Size of Overlay Area
47	IBFBP	; IDP to FIB of Installed System
65	IBCSH	; Current Size of LD (Hi)
66	IBCSL	; Current Size of LD (Lo)
67	IBMSH	; Max Size of LD (Hi)
70	IBMSL	; Max Size of LD (Lo)
	IBLEN=71	; Length of DIB

GLOSSARY

Associated: Space	AOS memory residing in the lowest 32Kw of the machine. Also known as GSMEM, accessing this memory does not require use of the MAP.
BLKQ:	The Process queue within AOS os implicitly blocked processes.
CANCH:	Otherwise known as the candidate chain, CANCH is a chain of all freed cache buffers. The AOS disk world scans this chain looking for disk blocks (elimenating physical disk I/O) and the Core Manager looks here in order to allocate pages during light memory contention.
Chunk:	A segment of memory within the AOS context. A chunk can be as small as 8 words or as large as 1,024.
Context:	An address somewhere in the AOS environment. It may be user space (the Primary Context), the Ghost (Secondary Context), or the operating system (the AOS context).
Disassocia-: ted Space	Another name for AOS GVMEM. In other words, memory space that is not associated with the core-resident Kernel of AOS.
DMA:	Direct Memory Access - a characteristic of <u>most</u> Data Channel devices.
D.O.B.:	Directory Overflow Block. An extra disk block enqueued to the directory when a "home" block overflows with FNB's. This is a sign of too many files in the directory or a poorly chosen hash frame size.
FAC:	File Access Control Block. Actually a variable length DDE, the FAC is allocated to hold ACL information for a given file. It is pointed to by the FIB IDP.
FDE:	File Data Element. A control block on the disk associated with file information. There are five kinds of DDE's: FAC's, FIB's, FLB's, FNB's and FUD's.

FIB:	File Information Block. A fixed length DDE on the used to hold the Filestatus information of a given file. It is pointed to by the FNB IDP and has IDP's to the other DDE's associated with the file.
F.I.F.O:	An acronym meanin First In First Out as distinct from L.I.F.O.
FLB:	File Link Block. Avariable length DDE use to hold the destination file pathname on a link entry.
FNB:	File Name Block. A variable length DDE used to hold the file name. It contains an IDP to the FIB.
FUD:	File User Data area. A fixed-length DDE created when as the file UDA. It is created with the ?CRUDA system call, and accessed via the ?RDUDA/?WRUDA system calls.
GSMEM:	General System Memory. A memory pool within the core-resident KERNEL of AOS - also known as Associated Space.
GVMEM:	General Virtual Memory. A memory pool within the AOS internal virtual memory area - also known as disassociated space. This memory is accessed by MAP B.
LDU:	Logical Disk Unit. One or more Physical disks [com
L.I.F.O.:	An acronym meanin Last In First Out as distinct from F.I.F.O.
MAP:	The Memory Allocation and Protection unit of the hardware. It is sometimes called the MMPU.
MBLKQ:	This an AOS internal queue of processes that have been explicitly blocked via the ?BLKPR system call.
MMPU:	The Memory Map and Protection Unit of the hardware. It is sometimes just called the MAP.
Nugget:	An "invisible" segment of disk space 8 words in length used to create DDE's. DDE's are often composed of several contiguous nuggets.

OVMCH:	The AOS internal chain of system overlay pages no longer in use by system call code. This chain is examined by the Core Manager during light memory contention.
P.I.O.:	Programmed I/O. I/O that is under direct control of the CPU rather than being handled completely by device controller.
RELQ:	The chain of Ready Eligble Processes. This chain resides in AOS Associated space.
RIB:	Random Index Block
Root:	The System's Root directory. This is the initial directory in the directory tree.

Performance Manual Index