Chapter 2

S/36 Memory Management

You hear conflicting stories when people discuss how the System/36 manages memory. Some people maintain that the System/36 is a swapping machine; others say it's a virtual machine. Many data processing managers believe System/36 memory architecture is simply a copy of the System/34 with minor changes; likewise, many programmers believe the System/36 limits tasks to 64 K because the System/34 has this limitation. Misconceptions arise from the lack of complete, understandable System/36 memory management information available to busy DP managers and programmers.

Although understanding the low-level details of S/36 memory management isn't essential, it helps you determine whether you have enough memory and whether you are using it effectively. And as you learn more about S/36 memory and how the system manages it, you can design S/36 programs that use memory efficiently.

Main Memory Organization

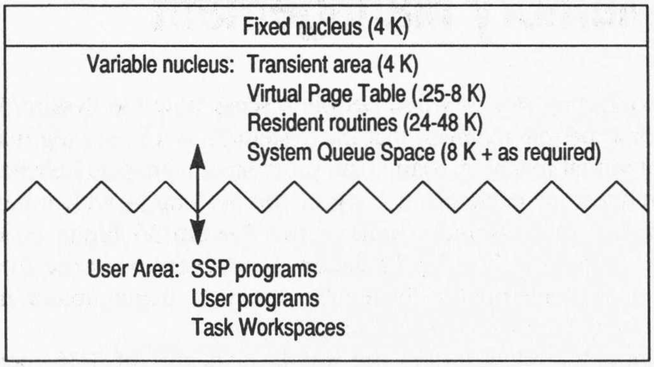
To develop a picture of S/36 memory, look at a diagram of S/36 main memory (Figure 2.1). Main memory is organized as eight-bit bytes and varies in size from 128 K to 7,168 K, depending on the machine model. Main memory consists of hundreds of integrated circuit "chips" and represents one of the most finite resources of the S/36. Figure 2.1 shows the three areas that comprise the contents of main memory: the fixed nucleus, the variable nucleus, and the user area.

The *fixed nucleus*, which occupies the first 4 K of main memory, contains variables and data structures needed by all components of the S/36's operating system, the System Support Program (SSP). The S/36's dual processors — the Main Storage Processor (MSP) and the Control Storage Processor (CSP) — also use the fixed nucleus to communicate with each other. Because the fixed nucleus is permanently set to the same size and content for all S/36 machines, a programmer or DP manager can do little to influence its effect on performance. However, an assembler language programmer can use the data stored in the fixed nucleus when writing special-purpose performance measurement tools (see MMETER Utility, chapter 14.)

The *variable nucleus* includes the transient area, virtual page table, resident routines, and system queue space. The transient area is 4 K of memory set aside for the very few SSP programs that must run in the variable nucleus. These programs are the task attach and detach, disk file open, diskette

Figure 2.1

Main Storage Contents


Fixed nucleus (4 K)
Variable nucleus: Transient area (4 K)
Virtual Page Table (.25-8 K)
Resident routines (24-48 K)
System Queue Space (8 K + as required)
User Area: SSP programs
User programs
Task Workspaces

Performance Tip

The SSP automatically queues up requests for the transient area, but a high volume of such requests can slow performance significantly by causing many jobs to wait for the transient area. You can reduce transient area contention by designing your applications to minimize new jobsteps (e.g., by using external program calls), thus reducing the need for task initiation/ termination and file open/close. Avoiding DDM situations that result in exceptions also helps minimize transient area contention (see Chapter 3).

open, and disk data management exception routines, which run infrequently enough so that contention for the transient area does not slow performance. The virtual page table is used by the S/36 virtual memory (VM) mechanism (described later) to keep the system operating even when memory is over-committed (i.e., when more programs are running than can fit in memory at one time). Resident routines are a few special SSP programs (disk data management and frequently used parts of workstation data management) that, for performance reasons, are always kept in main memory. System queue space (SQS) is a "pool" of memory set aside for dedicated use by SSP data structures needed to control the system.

Technical Note

Only one system program at a time can run in the transient area. Because file open/close, task attach/detach, and disk data management (DDM) exception handling all run in the transient area, SSP must perform these functions serially. For example, while a task such as // LOAD jobstep is being started, no files may be opened or closed. Similarly, when DDM exceptions occur (e.g., update of a key) no files may be opened or closed, or tasks initiated or terminated, until the exception is handled and the transient area becomes free.

The name "variable nucleus" implies the nature of this region: it varies in size with the amount of work performed by the system. The first three components of the variable nucleus don't actually change size while the machine is running; the amount of memory they occupy depends on the hardware and software configuration at IPL. Only the last area, system queue space (SQS), ebbs and flows with the varying system load. Because the first three components are "out of your hands," nothing more need be said about their function. On the other hand, your program design and scheduling *do* affect SQS, so a detailed knowledge of the SQS helps you make decisions that improve overall system performance. Later, we'll look at characteristics of SQS that are important from a performance standpoint.

The last, and usually largest, area of main memory is the *user area.* User programs, most SSP programs, file buffers, screen formats, and other objects reside here. One truism applied to computers in general, and the user area in particular, is: "You can't have too much main memory."

Effective memory management rests on your understanding of a few fundamental concepts: real memory, translated memory, and virtual memory. To grasp these ideas, let's look at main memory from a different angle.

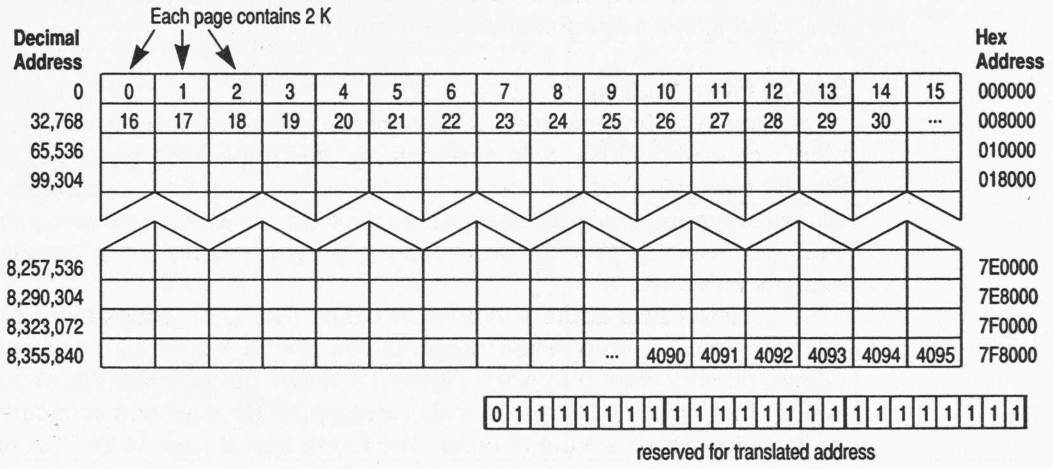
Performance Tip

Adding additional memory often drastically reduces interactive response times, making it the easiest and cheapest way to boost performance, especially given the availability of inexpensive used memory (see Chapter 5).

Memory Concepts

Figure 2.2 depicts the main memory address space for the S/36. Memory addressing is the practice of assigning to each location of computer memory a unique address, and using that address when referring to the contents of that location. The S/36 follows the popular convention of dividing memory into eight-bit bytes, each with a unique numeric address starting with zero. The number of bytes that byte-addressable memory may contain depends on the

Figure 2.2
System/36 Real Storage Organization



Performance Tip

The 5360 Model D is generally regarded as having a 7 MB memory limit. However, the *real address space* of the Model D supports an 8th MB. See Chapter 5 for information about adding an 8th MB of memory to your 5360 Model D.

size of the largest allowable address. On the S/36, a memory address is three bytes long, or 24 bits. The first bit of every address is set aside for a special purpose, leaving 23 bits to contain the address. The largest number that can be represented by 23 binary bits is 8,388,607, so the S/36 can theoretically have 8,388,608 bytes in its memory (remember, the first address is 0, not 1). These locations, or bytes, of memory, each with its unique address, make up *real memory* — all the memory that physically exists. The range of addresses, from 0 through 8,388,607, is the *real address space*: the entire set of unique memory locations available to the machine. The term real memory is used to differentiate between memory that is available on the hardware and memory that programs and programmers are *led to believe* is available (more on this type of memory later in the discussion of virtual memory).

Fragmentation

When you try to apply the real memory viewpoint in a multitasking system, problems arise, the worst of which is *fragmentation*. Figure 2.3 shows how this problem develops.

In Figure 2.3a, a hypothetical computer with 128 K of memory is running four programs that consume a total of 124 K. After programs B and D finish running, they release their memory, leaving two 24 K "holes" in the 128 K address space (Figure 2.3b). Later, the computer tries to run program E, which requires 32 K of memory (Figure 2.3c). Although a total of 48 K is available, the program is unable to run because available memory is split into two 24 K pieces. Program E must wait for either program A or C to end before it can obtain enough unbroken, or *contiguous*, memory. Because the usable address space is fragmented, program execution is delayed and perfectly good memory is wasted. Memory fragmentation worsens quickly in a busy computer system, causing system performance to drop off dramatically.

Solving Fragmentation

One solution to fragmentation is allowing programs to run in *noncontiguous* blocks of memory. The S/36 accomplishes this using addresses that do not directly correspond to real memory addresses but must be translated by special hardware into real addresses, hence the term *translated addressing*. Translated addresses appear to the executing program to represent contiguous memory locations.

Here's how address translation works. The S/36 groups bytes of real memory into 2 K units called *pages* (as you saw in Figure 2.2), each with a unique number from 0 to 4095. Figure 2.4 shows the program PROG A broken up into pages. Note that in main memory PROG A is not just fragmented — its logical pages are out of order. The fourth logical page of PROGA physically appears before the first logical page. Before such a fragmented program

Figure 2.3
How Memory Fragmentation Occurs

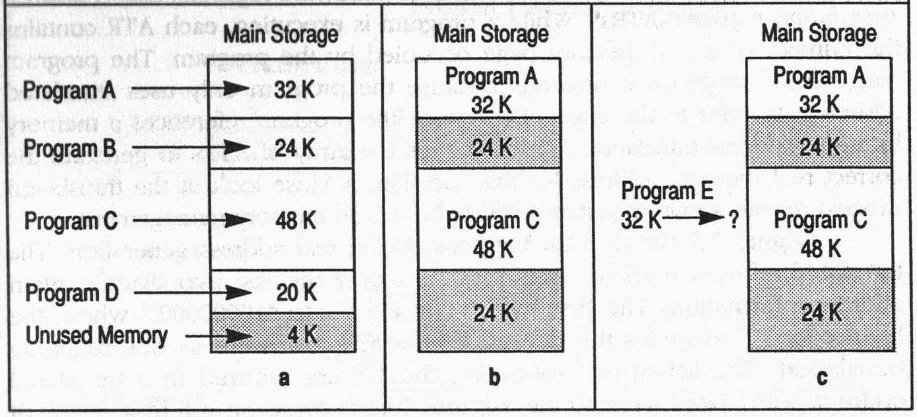
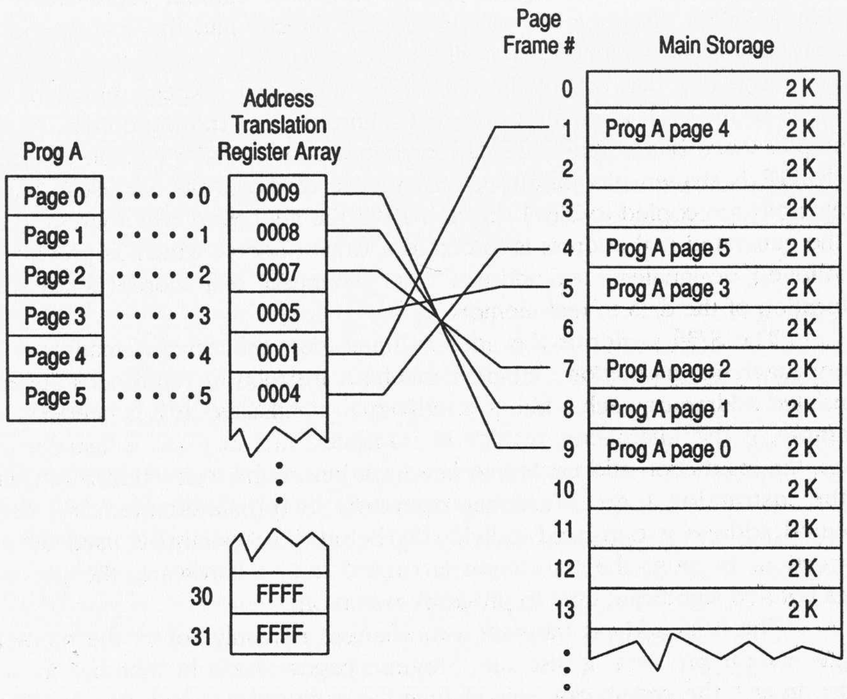


Figure 2.4
How the System/36 Solves Fragmentation



can run, a mechanism must rearrange the physical pages into their correct logical order.

The S/36 contains a special array of hardware registers, called *address translation registers* (ATRs). While a program is executing, each ATR contains the number of a real memory page occupied by the program. The program "sees" these pages as contiguous because the program only uses translated addresses to refer to the pages. Every time the program references a memory location, address translation hardware uses the array of ATRs to generate the correct real memory address for that location. A close look at the translation process reveals some important details about S/36 memory management.

Figure 2.5 shows what happens during real address generation. The translated address is three bytes long, but only the last two bytes contain address information. The first byte is always set to "10000000," where the high-order "1" identifies this address to the MSP hardware as one requiring translation. The seven bits following the "1" are ignored in a translated address. The sixteen remaining address bits provide an address space of 65,536 bytes, or 64 K. (This limit of using only two bytes for address information is the origin of the infamous 64 K region-size limitation.)

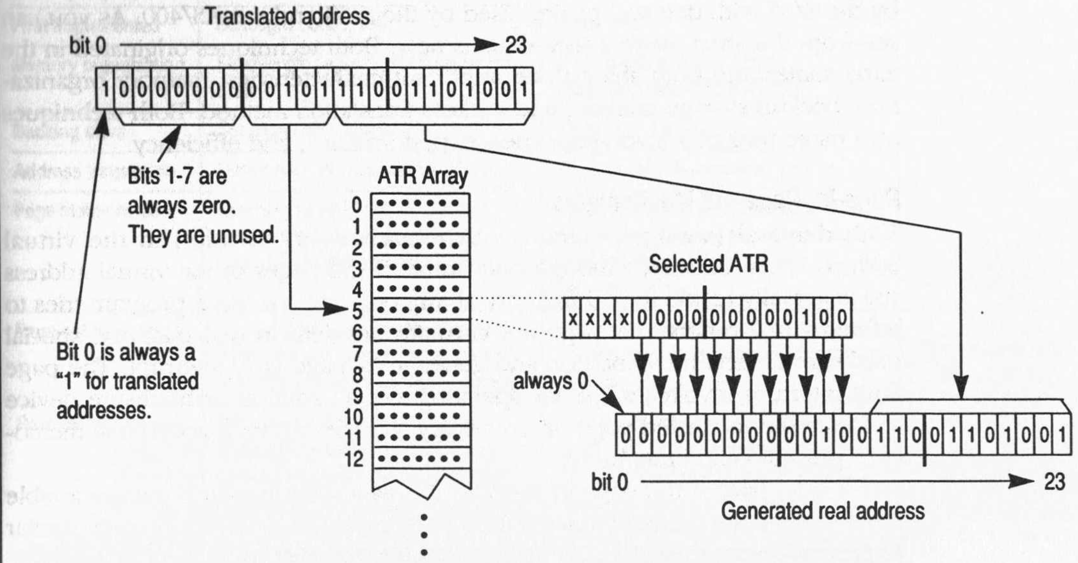
Because the last eleven bits of the translated address always fall within the boundaries of one logical page (the largest number represented by 11 bits is 2,047), these eleven bits are copied directly into the corresponding eleven bits of the generated real address.

The first five bits of the sixteen-bit translated address represent the number of the ATR containing the real memory page frame address. In the example, these bits contain "00101," or five, causing ATR #5 to be selected. Each ATR is sixteen bits long, but only twelve of those bits are used. Those twelve bits are copied to bits 1 through 12 of the generated real address. Bit 0 of the generated real address is forced to a value of zero, which as previously mentioned, designates a real address. This "generated" real address is the actual location of the data in real memory.

The S/36 performs the address translation process automatically for every machine instruction. When a machine instruction references several translated addresses, each address is individually translated as it is needed. For example, if the instruction resides in translated memory (as is usually the case), the instruction address is translated just before the instruction is fetched. If the instruction then references operands in translated memory, each operand address is translated individually before the operand is used by the instruction. Because the translation is carried out in hardware, the process does not add significant time to program execution.

The S/36 address translation mechanism not only solves the memory fragmentation problem, it also lets program pages reside in memory in any order. In fact, the system can even change the page order in memory, provided

Figure 2.5
Example of Generating a Real Address From a Translated Address



the ATR array for the moved pages is updated to reflect their new location. The useful ability to change page order without affecting the programs involved makes possible the next feature of S/36 memory management: virtual memory.

Virtual Memory

The fact that two levels of storage — primary and secondary — exist in most computer systems points up an ongoing compromise in computer technology. High-speed primary storage (such as the S/36's solid-state main memory) is too expensive and volatile for permanent data retention, so permanent information is stored on less expensive, but slower, secondary storage (usually disk). Primary storage contains only the data and programs the computer currently needs. However, the computer often works on several programs simultaneously — perhaps more than can fit in main memory at one time. When the number of currently executing programs exceeds the capacity of main memory, main memory is *overcommitted.* One way to handle overcommitment is to hide the true size of main memory from programs, letting them believe that there is much more memory than actually exists. The memory that programs use during execution — but that may not actually be available on the system — is called *virtual memory* (VM). The range of "imaginary" addresses is the *virtual address space.*

There are two popular ways to implement VM: segmented paging and demand paging. Figure 2.6 compares some features of segmented paging used by the S/36 with demand paging used by the S/38 and the AS/400. As you can see from the chart, neither technique is new. Both techniques originated in the early sixties and both share three important characteristics: memory organization, backup storage method, and address translation method. Both techniques also make tradeoffs involving expense, performance, and efficiency.

Page-in, Page-out Mechanisms

With demand paging, programs can reference any location in the virtual address space directly, although only some of the pages of the virtual address space actually reside in real memory at any one time. When a program tries to reference a location in a page not currently resident in real memory, special hardware detects the condition and generates a page fault interrupt. The page fault interrupt invokes a special operating system routine or hardware device to locate the requested page on secondary storage and read it into real memory, a process called *paging in.*

As part of the page-in process, the page fault handler updates a table used to generate real addresses during program execution — a process similar to S/36 address translation. To make room for the page to be read in, the page fault handler also may need to select a less important page in real memory and write it to secondary storage, a process called *paging out.* Usually, the paged-out page is chosen using an algorithm that finds the least-recently-used page in real memory.

The term *demand paging* comes from the fact that paging is driven by program references, or demands, to virtual memory. If a program never asks to "see" any location on a particular page, the page is never brought into real memory.

Segmented paging does not allow programs direct addressability to all locations in the virtual address space. Instead, programs have access only to a segment of virtual memory — 64 K in the case of the S/36. Instead of waiting for a program to reference a location in a nonresident page, the operating system keeps a list of pages currently being used by each executing program. When control switches from one program to another, the operating system compares the list of pages the next program requires with a list of pages currently in real memory (the virtual page table). If any pages are missing, the operating system retrieves them from secondary storage. If there are no free pages in real memory, the operating system writes some least-recently-used pages to secondary memory to free up enough pages for the next program to run.

The primary advantage of segmented paging — inexpensive imple-

Figure 2.6
Segmented and Demand Paging Comparison

	Segmented paging (S/36)	Demand paging (S/38 and AS/400)
First implemented	Burroughs B5000, 1961	Atlas, 1962
Memory organization	Fixed-length pages (2,048 bytes on the S/36)	Fixed-length pages (512 bytes on the S/38; 4096 bytes on the AS/400)
Backing store	Secondary disk storage	Secondary disk storage
Address translation	Dynamically with dedicated hardware.	Dynamically with dedicated hardware.
Page-in mechanism	Operating system knows program requirements and brings in required pages before giving program control.	Hardware detects program request for nonexistent page and generates a "page fault" to bring page in before task resumes execution.
Page-out mechanism	Pages for the lowest priority tasks are written out until enough pages are available for the program waiting for storage.	The least-recently-used page is written out and used to satisfy the page fault request.
Real memory usage	All pages for which a program has addressability must be in real storage before the program can run, regardless whether the program actually needs data in those pages now.	Only pages actually referenced by a program need be kept in real storage. Unused pages eventually are moved to secondary storage, freeing real storage for other programs.
Implementation	Mostly software. Address translation is assisted by special hardware.	Mostly hardware. Page faults and content management have special hardware assistance. Address translation is performed entirely in hardware.
Best features	Simplicity; lack of specialized hardware makes implementation less expensive; performance does not depend upon program behavior.	Hardware implementation improves both time and space efficiency; because only referenced pages are resident, memory utilization is good.
Worst features	Lack of hardware assistance means greater execution overhead; large programs tend to squander memory because unneeded pages are kept resident.	Hardware implementation is expensive; certain kinds of program behavior can cause repeated paging, known as "thrashing," which degrades performance.

mentation — comes from the fact that less complex address translation hardware is required. On the S/36, the address translation mechanism already is in place, making it easy to move pages in and out of real memory and rearrange them when necessary.

However, the inexpensive implementation exacts a price in performance. *All* the pages used by a program must be brought in before the program can resume execution, so some pages probably are not needed, and are wasted. Also, the special hardware used by demand paging to detect missing pages usually is much faster than the software-implemented virtual page table the S/36 uses for segmented paging.

On the S/36, this performance loss is mitigated to some extent, because the CSP can perform VM management chores while the MSP is working on user programs. But segmented paging also imposes a restriction on programmers: programs cannot exceed the size of one segment. On the S/36, the hardware-limited, 64 K segment size is uncomfortably small. Some systems other than the S/36 use a segmented paging approach that allows a program to use more than one segment, thus alleviating the S/36 restriction.

VM does, however, achieve its purpose. It theoretically can manage a virtual address space of 128 MB — 16 times larger than the maximum real address space of 8 MB. And it can manage this large virtual space efficiently. Many S/36 installations use external program calls to activate *all* of their frequently run programs for each user at the beginning of the day — hundreds of simultaneously active program segments amounting to 20 MB or more of VM. Because paging is much faster than reinitiating programs and reopening files, this technique eliminates redundant program initiation, reduces file open and close overhead, and improves response time dramatically.

Peculiarities of S/36 VM

The S/36 VM mechanism has a few unusual, and potentially confusing, twists. One common misconception is that the 64 K segment-size limitation, which also limits program size, limits *task* size. A task can contain one or more programs, each of which can be up to 64 K and must be executed individually. Because the number of programs that can be contained in a task on the S/36 is unlimited, the size of a task is also unlimited (up to the size of virtual address space).

rmance Tip

ing external program calls (EPC) itly reduces task initiation and file open overhead, oving interactive response time. nging your existig applications to EPC rather than L to interconnect grams is not very ult, making EPCs of the best tools ur kit for improvperformance (see Section III).

The S/36 contains a built-in external program call mechanism that lets one program invoke another separately compiled program, and then regain control when the called program returns. In addition, any number of called programs contained within a task may be simultaneously active. Active programs retain their internal state (values of variables and open files) from invocation to invocation.

Another oddity of the S/36 virtual implementation is the concept of *workspaces*, virtual segments that contain data instead of program code. Workspaces hold data buffers, screen formats, and various system-related tables and work areas, helping you get around the limitations of 64 K per program. An example of a workspace familiar to RPG programmers is the *disk file workspace*, which is created automatically when the 64 K segment for an RPG program has no room for disk file physical I/O buffers.

When a program needs to access data in a workspace, it calls on the operating system *map* facility, which gives the program addressability to the workspace by giving up some addressability to the program's virtual segment. Mapping, however, takes time and may result in paging activity, so the increased flexibility gained using workspaces is purchased with reduced performance.

A third unusual S/36 VM artifact is encountered only by installations that use a large amount of VM. On the S/36 the secondary storage used for paging is called the Task Work Area (TWA). The TWA is contained in a special system file called #SYSTASK that must reside on drive A1.

Initially, the maximum size for #SYSTASK is 6553 blocks (16 MB). This maximum is only about twice the maximum real memory size of 8 MB — not a very efficient overcommitment ratio. When the TWA is full, the SSP automatically extends the TWA by 400 blocks. When the TWA fills again, SSP doubles the extension to 800 blocks. Each time the TWA fills up, the size of the extension is doubled, allowing the TWA to grow to a very large size.

Unfortunately, each TWA extension requires contiguous space on drive A1. Drive A1 is also the default drive the system uses when allocating new files and work areas, which results in disk space fragmentation that may prevent the TWA from extending. Thus, the difficulty of obtaining disk space for paging can result in a much lower virtual address space limit than the 128 MB architectural maximum, unless the user takes steps to force TWA expansion before the A1 disk space becomes fragmented.

Performance Tip

One way to reduce TWA expansion problems is to "pre-allocate" the TWA by activating all your programs in advance — usually immediately after IPL (see Chapter 7).

System Queue Space

Now that you understand real, translated, and virtual memory, you can appreciate the effort undertaken by the S/36 to administer memory usage efficiently. Although address translation and segmented paging improve memory use by effectively reusing a limited resource, not everything in real memory can be moved about with abandon. Only objects in the user area accommodate this manipulation. A certain amount of real memory — the fixed and variable nuclei — must remain resident and can be accessed only through real addressing.

All of the fixed nucleus and most of the variable nucleus is static (unmoving) — beyond your control. As mentioned earlier, programming techniques directly affect only one part of the variable nucleus: system queue space. Knowing how your application design decisions impact SQS use helps you make educated compromises between performance and simplicity.

SQS is an expandable "pool" of memory used by the SSP and the CSP to hold dynamically allocated data structures, called *control blocks*, critical to the operation of the system. Once a control block is created in SQS, it remains resident in real memory at the same location until explicitly destroyed. Because each control block must occupy contiguous memory locations, SQS can become fragmented.

Control blocks range in size from 16 bytes to 2,048 bytes, in 16-byte increments. They can be categorized by their life spans: short, medium, and long. A short-lived control block's life span is only a few milliseconds. The SSP creates short-lived control blocks for the duration of certain brief chores (e.g., a disk file operation) and destroys them when the chore is complete. Medium-lived control blocks last a relatively long time — for the duration of a job, for instance. Long-lived control blocks (usually created when the system is started) are the very few that become permanent until the next IPL.

The system keeps a modest reserve of SQS available (about 2000 to 4000 bytes) to satisfy most control block creation requests quickly. When this reserve is consumed, the system takes a 2 K page away from the user area and adds it to SQS. (Because a control block cannot be larger than 2 K bytes, the newly acquired page can be obtained from anywhere in real memory.) The system continues to take 2 K pages from the user area as needed. When more than about 4000 bytes accumulates in the SQS reserve area (due to control blocks being freed), the system returns a 2 K page to the user area. Thus, the logical "boundary" between SQS and the user area fluctuates constantly to meet the needs of the system.

Of the three classes of control blocks, only one is of concern to you. Short-lived control blocks have minimal impact on system performance, and long-lived control blocks are beyond your control. Only medium-lived control blocks have a controllable impact on system performance; most medium-lived control blocks are a direct result of the kinds of programs you design. The table in Figure 2.7 summarizes the space requirements for the most common control blocks and the program activities that create them.

The table also will help you determine the amount of SQS a given program or device needs to run. Computing the SQS requirements for an entire job mix lets you estimate the total amount of real memory that will be dedicated to SQS, and therefore will be unavailable in the user area. For example, an interactive job with ten indexed files, a printed report, and five subprograms requires 9,088 bytes of SQS:

- 192 bytes for the workstation session control block
- 256 bytes for the job control block
- 96 bytes for the task control block
- 320 bytes for the active programs (64 bytes each)
- 96 bytes for one level of subprogram invocation
- 64 bytes for a disk file workspace
- 688 bytes for the opened print file
- 1,600 bytes for disk file VTOC entries (160 bytes each)
- 1,680 bytes for other file-related control block (file specification block, file buffer block, disk buffer block, allocation queue element, record queue

Figure 2.7
System Queue Space Requirements for Common Control Blocks

SSP entity	Control Block	Total SQS Bytes Used
Each local workstation session	Terminal Unit Block (192 bytes)	192
Each printer	Printer Unit Block (96 bytes)	96
Each remote device	RWS Device Unit Block (80 bytes)	80
Each job	Job Control Block (256 bytes)	256
Each task	Task Block (96 bytes)	96
Each active program or subprogram	Program Block (64 bytes)	64
Each invoked program or subprogram	Request Block (64-2048 bytes)	96 (avg)
Each workspace	Storage Block (64 bytes)	64
Each user of an opened file	File Specification Block (64 bytes) File Buffer Block (24 bytes) Disk Buffer Block (16 bytes)	104
Additional overhead for first user to open a file or use a library	Format-1, or VTOC entry (160 bytes)	160
Additional overhead for each user of a shared file	Allocation Queue Element (32 bytes) Record Queue Block (16 bytes)	48
Additional overhead for each user of an indexed file	Index Control Block (16 bytes)	16
Each storage indexed file storage index	Depends on the size of a storage index (the storage for a file is shared by all users of the file)	varies
Each opened print file being spooled	Printer Specification Block (64 bytes) Spool File Descriptor (112 bytes) Spool intercept buffer (256-2048 bytes)	688 (avg)
Each active spool writer	Writer Descriptor Block (48 bytes) Task Block (96 bytes) Spool print buffer (256-2048 bytes)	1168 (avg)

block, and index control block) , and
- 4,096 bytes for storage indexes (estimated)

If you plan to run the program from nine workstations simultaneously, the additional eight workstations require 3,392 bytes of SQS each (the VTOC control blocks and storage indexes are counted only for the first user), resulting in a grand total of 36,224 bytes of SQS. Remember that SQS use reduces the amount of memory available in the user area for virtual use, thereby increasing the "swap rate" (level of paging activity), and possibly degrading system performance. If you run these programs on a 512 K system, you might find installing another 256 K memory board a cost-effective way of maintaining acceptable response time.

Considering all aspects of S/36 memory management, you can see why misconceptions abound. But the S/36 loses its mystique once you master the secrets of its memory. You can use this knowledge to help plan future expansion of your S/36 and to evaluate its place in the midrange system market. Careful evaluation of memory requirements lets you predict the effect of additional memory more accurately. And, of course, the better you understand your S/36, the better you can take advantage of its features to improve performance.