

Course Number: EECIE18-S3303

Course Name: Operating System – Special-Purpose Systems

Textbook: Stallings W., "Operating Systems: Internals and design principles", 7th Edition, 2012, Pearson Education Limited, ISBN: 978-0-13-230998-1.

Lecturer: Dr. Sahar AL-talib

Operation of Two-Level Memory

The upper-level memory (M1) is smaller, faster, and more expensive (per bit) than the lower-level memory (M2). M1 is used as temporary store for part of the contents of the larger M2. When a memory reference is made, an attempt is made to access the item in M1. If this succeeds, then a quick access is made. If not, then a block of memory locations is copied from M2 to M1 and the access then takes place via M1.

Because of locality, once a block is brought into M1, there should be a number of accesses to locations in that block, resulting in fast overall service.

To express the average time to access an item, we must consider not only the speeds of the two levels of memory but also the probability that a given reference can be found in M1. We have

$$T_s = H \times T_1 + (1 - H) \times (T_1 + T_2)$$
$$T_1 + (1 - H) \times T_2$$

Where

T_s = average (system) access time

T_1 = average time of M1 (e.g., cache, disk, cache)

T_2 = average time of M2 (e.g., main memory, disk)

H = hit ratio (fraction of time reference is found in M1)

Figure 1 shows average access time as a function of hit ratio. As can be seen, for high percentage of hits, the average total access time is much closer to that of M1 than M2.

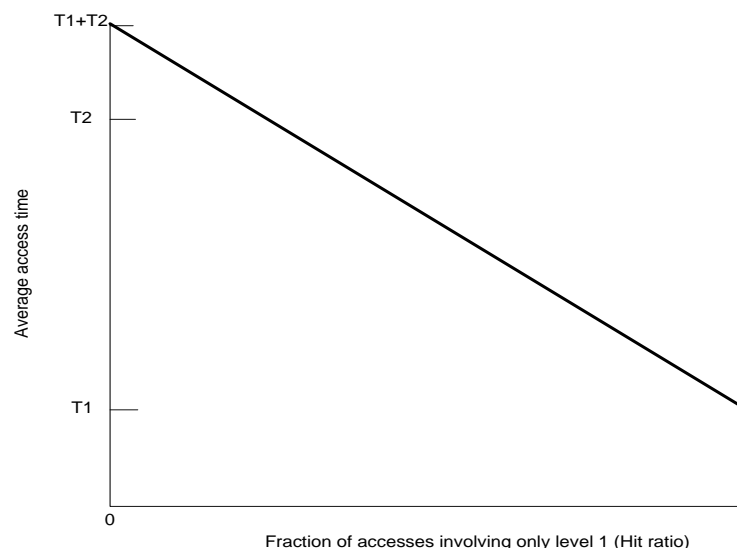


Figure: Performance of a Simple Two-level Memory

Performance

Let us look at some of the parameters relevant to an assessment of a two-level memory mechanism. First consider cost. We have

$$C_s = \frac{C_1 S_1 + C_2 S_2}{S_1 + S_2}$$

Where

C_s = average cost per bit for the combined two-level memory

C_1 = average cost per bit of upper-level memory M1

C_2 = average cost per bit of lower-level memory M2

S_1 = size of M1

S_2 = size of M2

We would like $C_s \approx C_2$. Given that $C_1 \gg C_2$, this requires $S_1 \ll S_2$. Figure 2 shows the relationship.

Next, consider access time. For a two-level memory to provide a significant performance improvement, we need to have T_s approximately equal to $T_1 T_s \approx T_1$.

Given that T_1 is much less than T_2 , $T_s \gg T_1$, a hit ratio of close to 1 is needed.

So we would like M1 to be small to hold down cost, and large to improve the hit ratio and therefore the performance. Is there a size of M1 that satisfies both requirements to a reasonable extent? We can answer this question with a series of subquestions:

- What value of hit ratio is needed to satisfy the performance requirement?
- What size of M1 will assure the needed hit ratio?
- Does this size satisfy the cost requirement?

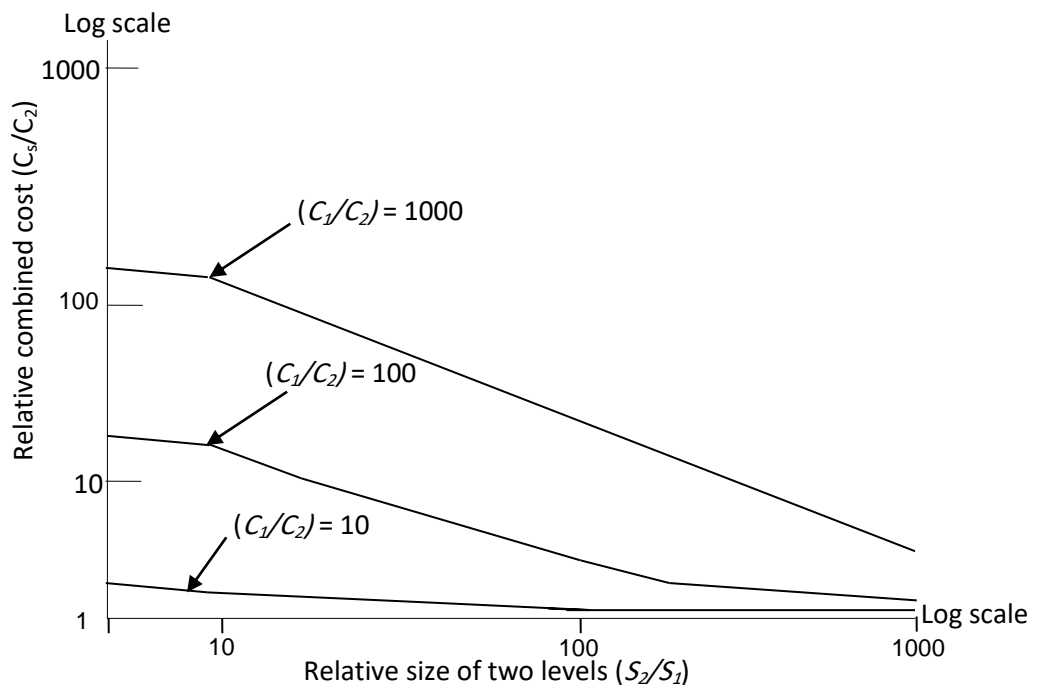


Figure 2: Relationship of Average Memory Cost to Relative Memory Size for a Two-Level Memory

Virtual memory is a facility that allows programs to address memory from a logical point of view, without regard to the amount of main memory physically available. Virtual memory was conceived to meet the requirement of having multiple user jobs reside in main memory concurrently, so that there would not be hiatus between the executions of successive processes while one process was written out to secondary store and the successor process was read in. because processes vary in size, if the processor switches among a number of processes it is difficult to pack them compactly into main memory. Paging systems were introduced, which allow processes to be comprised of a number of fixed-size blocks, called pages. A program references a word by means of a virtual address consisting of a page number and an offset within the page. Each page of a process may be located anywhere in main memory. The paging system provides for a dynamic mapping between the virtual address used in the program and a real address, or physical address, in the main memory.

With dynamic mapping hardware available, the next logical step was to eliminate the requirement that all pages of a process reside in memory simultaneously. All the pages of a process are maintained on disk. When a process is executing, some of its pages are in main memory. If reference is made to a page that is not in main memory, the memory management hardware detects this and arranges for the missing page to be loaded. Such a scheme is referred to as virtual memory and is depicted in Figure 3.

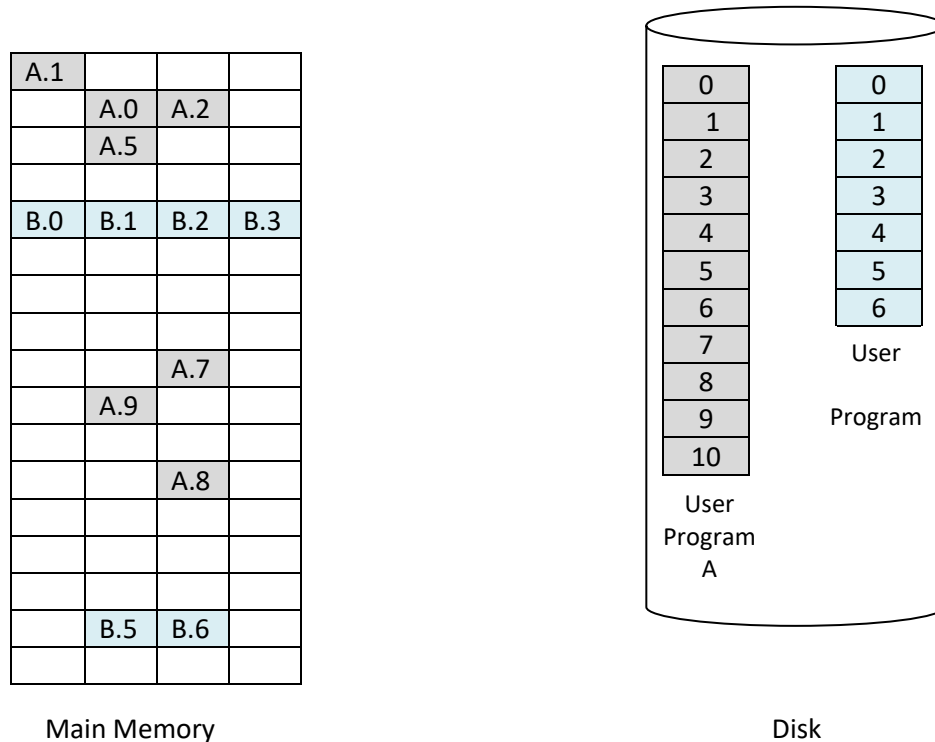


Figure 3: Virtual Memory Concepts

Main memory

Main memory consists of a number of fixed-length frames, each equal to the size of a page.
For a program to execute, some or all of its pages must be in main memory.

Disk

Secondary memory (disk) can hold many fixed-length pages. A user program consists of some number of pages. Pages for all programs plus the operating system are on disk, as are files.

Virtual Machines and Virtualizing

Traditionally, applications have run directly on an OS on a server. Each PC or server would run only one OS at a time. Thus, the vendor had to rewrite parts of its applications for each OS/platform they would run on. An effective strategy for dealing with this problem is known as virtualization. Virtualization technology enables a single PC or server to simultaneously run multiple operating systems or multiple sessions of a single OS. A machine with virtualization can host numerous applications, including those that run on different operating systems, on a single platform. In essence, the host operating system can support a number of virtual machines (VM), each of which has the characteristics of a particular hardware platform.

The VM approach is becoming a common way for business and individuals to deal with legacy applications and to optimize their hardware usage by maximizing the number of kinds of applications that a single computer can handle.

Commercial VM offerings by companies such as VMware and Microsoft are widely used. In addition to their use in server environments, these VM technologies also are used in desktop environments to run multiple operating systems, typically Windows and Linux.

The specific architecture of the VM approach varies among vendors. Figure 4 shows a typical arrangement. The virtual machine monitor (VMM), or hypervisor, runs on top of (or is incorporated into) the host OS. The VMM supports VMs, which are emulated hardware devices. Each VM runs a separate OS. The VMM handles each operating system's communications with the processor, the storage medium, and the network. To execute programs, the VMM hands off the processor control to a virtual OS on a VM. Most VMs use virtualized network connections to communicate with one another, when such communication is needed. Key to the success of this approach is that the VMM provides a layer between software environments and the underlying hardware and host OS that is programmable, transparent to the software above it, and makes efficient use of the hardware below it.

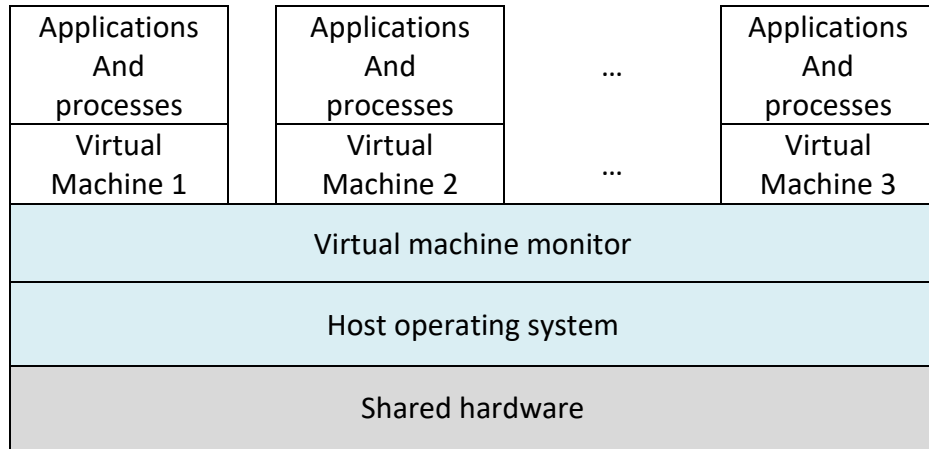


Figure 4: Virtual Memory Concept

OS Design considerations for Multiprocessor and Multicore

Symmetric Multiprocessor (SMP) OS considerations

An SMP operating system manages processor and other computer resources so that the user may view the system in the same fashion as a multiprogramming uniprocessor system. A user may construct applications that use multiple processes or multiple threads within processes without regard to whether a single processor or multiple processors will be available.

The key design issues include the following:

- **Simultaneous concurrent processes or threads** (متزامن في وقت واحد مترام processes or threads) (A thread of execution is the smallest sequence of programmed instructions that can be managed independently by an operating system scheduler. A thread is a light-weight process).: Kernel routines need to be reentrant (متزامن مع عدة طلبات بنفس الوقت) to allow several processors to execute the same kernel code simultaneously.
- **Scheduling:** Any processor may perform scheduling, which complicates the task of enforcing a scheduling policy and assuring that corruption of the scheduler data structures is avoided. If kernel-level multithreading is used, then the opportunity exists to schedule multiple threads from the same process simultaneously on multiple processors.
- **Synchronization:** it is a facility that enforces mutual exclusion and event ordering.
- **Memory management:** the paging mechanism on different processors must be coordinated to enforce consistency when several processors share a page or segment and to decide on page replacement. The reuse of physical pages is the biggest problem of concern; that is, it must be guaranteed that a physical page can no longer be accessed with its old contents before the page is put to a new use.

- **Reliability and fault tolerance:** The OS should provide graceful degradation in the face of processor failure. The scheduler and other portions of the OS must recognize the loss of a processor and restructure management tables accordingly.

Multicore OS Considerations

Current multicore vendors offer systems with up to eight cores on a single chip. The design challenge for a many-core multicore system is to efficiently harness the multicore processing power and intelligently manage the substantially on-chip resources efficiently. A central concern is how to match the inherent parallelism of a many-core system with the performance requirements of applications. The potential for parallelism in fact exists at three levels in contemporary multicore system.

1. There is H/W parallelism within each core processor, known as instruction level parallelism, which may or may not be exploited by application programmers and compilers.
2. There is the potential for multiprogramming and multithreaded execution within each processor.
3. There is the potential for a single application to execute in concurrent processes or threads across multiple cores.

Without strong and effective OS support for the last two types of parallelism just mentioned, hardware resources will not be efficiently used.