NTNU – Trondheim
Norwegian University of
Science and Technology

Compendium

TDT4186 Operating systems

Curriculum:

- William Stallings Operating Systems 8th ed. chapter 1-12 and Appendix A.
- The slides from the course.

Web content:

`http://williamstallings.com/OperatingSystems/`
`http://williamstallings.com/OS-Animation/Animations.html`

February 19, 2018

The course will give students an understanding of the tasks that an operating system solves, how tasks are solved and give an understanding of the use characteristics of an operating system. The themes are the kernel of the operating system, processes, threads, synchronization, time-sharing, memory management, file systems, I/O, deadlock management, multi-processor systems and security.

Contents

1 Computer System Overview

Processor Controls the operation of the computer and performs its data processing functions.

Main memory Stores data and programs, typically volatile

I/O modules Move data between the computer and its external environment.

System bus Provides for communication among processors, main memory and I/O modules

MAR Memory address register, specifies the address in memory for the next read or write

MBR Memory buffer register, contains the data to be written into memory or which receives the data read from memory.

GPU Graphical processing units, efficient computation on arrays of data using Single-Instruction Mulitple Data techniques.

DSPs Digital Signal Processors, for dealing with streaming signals such as audio or video (embedded in I/O devices)

SoC System on Chip. Not just the CPU and caches are on the same chip, but also many of the other components such as DSPs, GPUs, I/O devices and main memory.

Spatial locality Refers to the tendency of execution to involve a number of memory locations that are clustered.

Temporal locality Refers to the tendency for a processor to access memory locations that have been used recently.

QPI QuickPath Interconnect. Point-to-point link electrical interconnect specification. Enables high-speed communications among connected processor chips.

1.1 Categories of processor actions

The four categories of processor actions:

- Processor-memory: Data may be transferred from processor to memory or reverse.
- Processor-I/O: Data may be transferred to or from a peripheral device (I/O)
- Data processing: The processor may perform some arithmetic or logic operation on data
- Control: An instruction may specify that the sequence of execution be altered (Branching to another instruction in memory by changing the value in the program counter).

1.2 SMP

Symmetric Multiprocessor. Stand-alone computer system with the following characteristics:

- Two or more similar processor of comparable capability.
- Processors are interconnected and share the same main memory and I/O. Memory access time approximately the same.
- All processor share access to I/O devices.
- All processors can perform the same functions (symmetric)
- The system is controlled by an integrated operating system the provides interaction between processors and their programs at the job, task, file and data element levels.

1.3 Review Questions

1.1 List briefly and define the four main elements of a computer.

1.2 Define the two main categories of processor registers.

1.3 In general terms, what are the four distinct actions that a machine instruction can specify?

1.4 What is an interrupt?

1.5 How are multiple interrupts dealt with?

1.6 What characteristics distinguish the various elements of a memory hierarchy?

1.7 What is cache memory?

1.8 What is the difference between a multiprocessor and a multicore system?

1.9 What is the distinction between spatial locality and temporal locality?

1.10 In general, what are the strategies for exploiting spatial and temporal locality?

2 Operating System Overview

ISA Instruction set architecture. The ISA defines the repertoire of machine language instructions that a computer can follow.

ABI Application binary interface. The ABI defines a standard for binary portability across programs. The ABI defines the system call interface to the operating system and the hardware resources and services abailable in a system through the user ISA.

API Application programming interface. The API gives a programm access to the hardware resources and services available in a system through the user ISA supplemented with high-level language library calls.

JCL Job control language. Special type of programming language used to provide instructions to the monitor.

Types of operating systems:

- Serial processing: Programs loaded in serial via an input device.
- Simple batch systems: Use of a monitor, jobs are batched together sequentially. Each job are programmed to batch back to the monitor when processing is completed, at which point the monitor automatically begins loading the next program.
- Multiprogrammed batch systems: Switching between multiple programs in the processor with the use of interrupts and DMA.
- Time-sharing systems: Same as multiprogrammed batch systems, but is also able to handle multiple interactive jobs. Process time is share among multiple users.

OS five principal memory management responsibilities:

- Process isolation: The OS must prevent independent processes from interfering with each others memory.
- Automatic allocation and management: Programs should be dynamically allocated across the memory hierarchy as required. Should be transparent to the programmer.
- Support of modular programming: Programmers should be able to define program modules, and to create, destroy, and alter the size of modules dynamically.

- Protection and access control: Sharing of memory can be favorable when different processes need access to the same data. In other cases it threatens the integrity of programs and the OS itself.
- Long-term storage: Many programs require to store data for a long time, even after the computer has been powered down.

2.1 Review Questions

2.1 What are the three objectives of an OS design?

2.2 What is the kernel of an OS?

2.3 What is multiprogramming?

2.4 What is a process?

2.5 How is the execution context of a process used by the OS?

2.6 List and briefly explain five storage management responsibilities of a typical OS.

2.7 Describe the round-robin scheduling technique.

2.8 Explain the difference between a monolithic kernel and a microkernel.

2.9 What is multithreading?

2.10 List det key design issues for an SMP operating system.

3 Process Description and Control

Process A process has several definitions:

- A program in execution
- The instance of a program running on a computer
- The entity that can be assigned to and executed on a processor
- The unit of activity characterized by the execution of a sequence of instructions, a current state, and an associate set of system resources

Dispatcher Responsible for saving the current process context and load the context of the process to run next.

Preemption Reclaiming of a resource from a process before the process has finished using it.

Process Image Collection of program code, process data, process stack and process control block (attributes)

PSW Program status word. Containts status information such as condition codes etc.

While a program is executing, this process can be uniquely characterized by the following elements:

- Identifier: A unique identifier associated with the process.
- State: If the process is running, it is in the running state.
- Priority: Priority level relative to other processes.
- Program counter: The address of the next instruction in the program to be executed.
- Memory pointers: Include pointers to the program code and data associated with this process, plus any memory blocks shared with other processes.
- Context data: Data that are present in the registers while the process is executed.
- I/O status information: Outstanding I/O requests and I/O devices assigned to this process, a list of files in use by the process and so on.
- Accounting information: Amount of processor time and clock time used, time limits, account numbers and so on.

The information stored in the preceding list is stored in a process control block. This is needed to be able to interrupt the process and later resume execution as if nothing happened. Key tool for multiprocessing. The typical elements of a process control block can be seen in Figure 3.1

3.1 Process modeling

The process image is the collection of program, data, stack and attributes (Figure 3.2).

3.2 Creating a process

Steps of creating a process:

- Assign a unique process identifier to the new process.
- Allocate space for the process (all elements of the process image).
- Initialize the process control block.
- Set the appropriate linkages (put in state queue etc.).
- Create or expand other data structures (e.g. profiling).

3.3 Review questions

3.1 What is an instruction trace?

3.2 What common events lead to the creation of a process?

3.3 For the five-state processing model of Figure 3.3, briefly define each state.

3.4 What does it mean to preempt a process?

3.5 What is swapping and what is its purpose?

3.6 Why can a process model have two blocked states?

3.7 List four characteristics of a suspended process.

3.8 For what types of entities does the OS maintain tables of information for management purposes?

3.9 List three general categories of information in a process control block.

3.10 Why are two modes (user and kernel) needed?

3.11 What are the steps performed by an OS to create a new process?

3.12 What is the difference between an interrupt and a trap?

3.13 Give three examples of an interrupt.

3.14 What is the difference between a mode switch and a process switch?

3.15 List four general categories of information in a process image.

Process Identification

Identifiers
Numeric identifiers that may be stored with the process control block include

- Identifier of this process.
- Identifier of the process that created this process (parent process).
- User identifier.

Processor State Information

User-Visible Registers
A user-visible register is one that may be referenced by means of the machine language that the processor executes while in user mode. Typically, there are from 8 to 32 of these registers, although some RISC implementations have over 100.

Control and Status Registers
These are a variety of processor registers that are employed to control the operation of the processor. These include

- **Program counter:** Contains the address of the next instruction to be fetched.
- **Condition codes:** Result of the most recent arithmetic or logical operation (e.g., sign, zero, carry, equal, overflow).
- **Status information:** Includes interrupt enabled/disabled flags, execution mode.

Stack Pointers
Each process has one or more last-in-first-out (LIFO) system stacks associated with it. A stack is used to store parameters and calling addresses for procedure and system calls. The stack pointer points to the top of the stack.

Process Control Information

Scheduling and State Information
This is information that is needed by the operating system to perform its scheduling function. Typical items of information:

- **Process state:** Defines the readiness of the process to be scheduled for execution (e.g., running, ready, waiting, halted).
- **Priority:** One or more fields may be used to describe the scheduling priority of the process. In some systems, several values are required (e.g., default, current, highest-allowable).
- **Scheduling-related information:** This will depend on the scheduling algorithm used. Examples are the amount of time that the process has been waiting and the amount of time that the process executed the last time it was running.
- **Event:** Identity of event the process is awaiting before it can be resumed.

Data Structuring
A process may be linked to other process in a queue, ring, or some other structure. For example, all processes in a waiting state for a particular priority level may be linked in a queue. A process may exhibit a parent–child (creator–created) relationship with another process. The process control block may contain pointers to other processes to support these structures.

Interprocess Communication
Various flags, signals, and messages may be associated with communication between two independent processes. Some or all of this information may be maintained in the process control block.

Process Privileges
Processes are granted privileges in terms of the memory that may be accessed and the types of instructions that may be executed. In addition, privileges may apply to the use of system utilities and services.

Memory Management
This section may include pointers to segment and/or page tables that describe the virtual memory assigned to this process.

Resource Ownership and Utilization
Resources controlled by the process may be indicated, such as opened files. A history of utilization of the processor or other resources may also be included; this information may be needed by the scheduler.

Figure 3.1: Typical elements of a Process Control Block

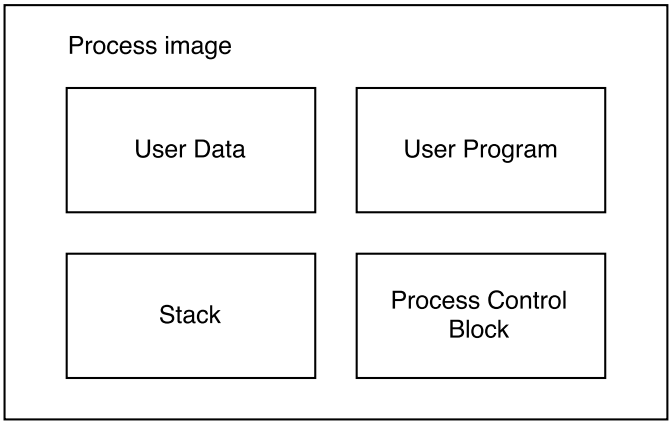


Figure 3.2: Elements of a process image

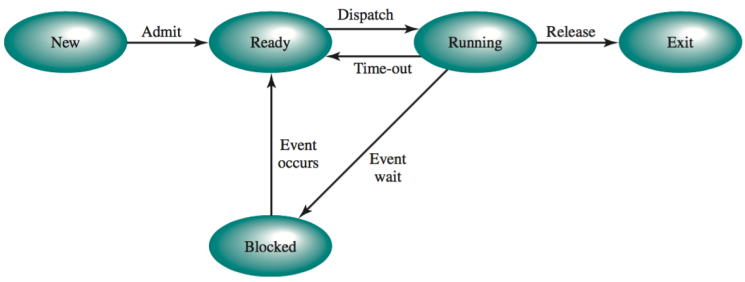


Figure 3.3: Five-State Process Model

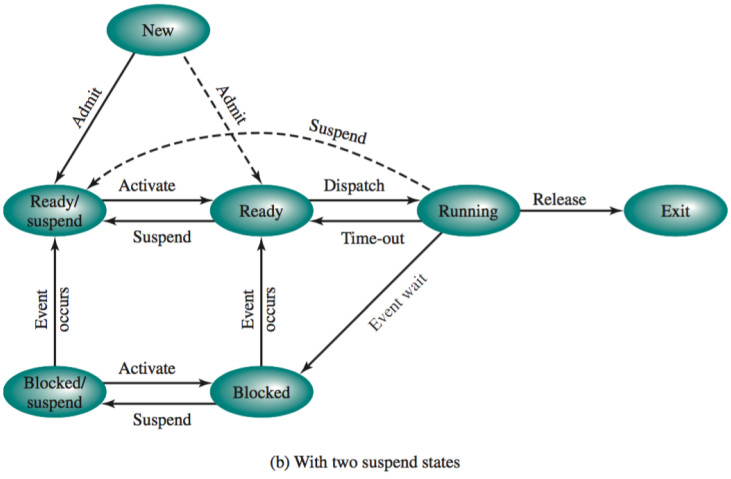


Figure 3.4: Process State Transition Diagram with Suspend States

4 Threads

ULT User-Level Threads. All the work of thread management is done by the application and the kernel is not aware of the existence of threads.

KLT Kernel-Level Threads. All the work of a thread management is done by the kernel. There is no thread management code in the application level, simply and API to the kernel thread facility.

Jacketing Way of overcoming the problem of blocking threads in ULTs. Convert a blocking system call into a non-blocking system call. For example, instead of directly calling a system I/O routine, a thread calls an application level jacket routine. The jacket routine checks if the I/O is busy. If the I/O is busy the thread enters blocked state and passes control through the threads library to another thread. When this thread is later given control again, the jacket routine checks the I/O device again.

Some operating system distinguish the concepts of process (related to resource ownership) and thread (related to program execution). Process:

- A virtual address space that holds the process image.
- Protected access to processors, other processes, files and I/O resources.

Threads:

- A thread execution state.
- A saved thread context when not running; one way to view a thread is an independent program counter operating within a process.
- An execution stack.
- Some per-thread static storage for local variables
- Access to the memory and resources of its process, shared with all other threads in that process.

This approach may lead to improved efficiency and coding convenience. In a multi-threaded system, multiple concurrent threads may be defined within a single process. This may be done using either user-level threads or kernel-level threads.

- User-level threads are unknown to the OS and are created and managed by a threads library that runs in the user space of a process. User-level threads are very efficient because a mode switch is not required to switch from one thread to another. However, only a single user-level thread within a process can execute at a time, and if one thread blocks, the entire process is blocked.
- Kernel-level threads are threads within a process that are maintained by the kernel. Because they are recognized by the kernel, multiple threads can within the same process can execute in parallel on a multiprocessor. Blocking of a thread does not block the entire process. However, a mode switch is required to switch from on thread to another, and this takes some time.

4.1 Review questions

4.1 Figure 3.1 lists typical elements found in a process control block for an unthreaded OS. Of these, which should belong to a thread control block, and which should belong to a process control block for a multithreaded system?

4.2 List reasons why a mode switch between threads may be cheaper than a mode switch between processes.

4.3 What are the two separate and potentially independent characteristics embodied in the concept of process?

4.4 Give four general examples of the use of threads in a single-user multiprocessing system.

4.5 What resources are typically shared by all of the threads of a process?

4.6 List three advantages of ULTs over KLTs.

4.7 List two advantages of KLTs over ULTs.

4.8 Define jacketing.

5 Concurrency: Mutual Exclusion and Synchronization

Multiprogramming The management of multiple processes within a uniprocessor system.

Multiprocessing The management of multiple processes within a multiprocessor.

Distributed processing The management of multiple processes executing on multiple distributed computer systems. The recent proliferation of clusters is a prime example of this type of system.

Atomic operation A function or action implemented as a sequence of one or more instructions that appear to be indivisible; that is, no other process can see an intermediate or interrupt the operation. The sequence of instruction is guaranteed to execute as a group, or not execute at all, having no visible effect on system state. Atomicity guarantees isolation from concurrent processes.

Critical section A section of code within a process that requires access to shared resources and that must not be executed while another process is in a corresponding section of code.

Deadlock A situation in which two or more processes are unable to proceed because each is waiting for the others to do something.

Livelock A situation in which two or more processes continuously change their states in response to changes in the other process(es) without doing any useful work.

Mutual exclusion The requirement that when one process is in a critical section that accesses shared resources, no other process may be in a critical section that accesses any of those shared resources.

Race condition A situation in which multiple threads or processes read and write a shared data item and the final result depends on the relative timing of their execution.

Starvation A situation in which a runnable process is overlooked indefinitely by the

scheduler; although it is able to proceed, it is never chosen.

Semaphore An integer value for signaling among processes. Only three operations may be performed on a semaphore, all of which are atomic: initialize, decrement, and increment. The decrement operation may result in the blocking of a process, and the increment option may result in the unblocking of a process. Also known as counting semaphore or general semaphore.

Binary semaphore A semaphore that takes on only the values 0 and 1.

Mutex Mutual exclusion lock. Similar to a binary semaphore. A key difference between the two is that the process that locks the mutex (sets the value to zero) must be the one to unlock it.

Monitor A programming language construct that encapsulates variables, access procedures, and initialization code within an abstract data type. The monitor's variable may only be accessed via its access procedures and only one process may be actively accessing the monitor at any time. The access procedures are critical sections. A monitor may have a queue of processes that are waiting to access it.

Concurrency arises in three different contexts:

- Multiple applications: Multiprogramming was invented to allow processing time to be dynamically shared among a number of active applications.
- Structured applications: As an extension of the principles of modular design and structured programming, some applications can be effectively programmed as a set of concurrent processes.
- Operating system structure: The same structuring advantages apply to systems programs, and we have seen that operating systems are themselves often implemented as a set of processes or threads.

Design challenges for OS raised by the existence of concurrency:

- The OS must be able to keep track of the various processes.
- The OS must allocate and deallocate resources for each active process. Multiple processes might want access to the same resource (processor, memory, files, I/O) simultaneously.
- The OS must protect the data and physical resources of each process against unintended interference by other processes.
- The functioning of a process, and the output it produces, must be independent of the

speed at which its execution is carried out relative to the speed of other concurrent processes.

5.1 Review questions

5.1 List four design issues for which the concept of concurrency is relevant.

5.2 What are three contexts in which concurrency arises?

5.3 What is the basic requirement for the execution of concurrent processes?

5.4 List three degrees of awareness between processes and briefly define each.

5.5 What is the distinction between competing processes and cooperating processes?

5.6 List three control problems associated with competing processes and briefly define each.

5.7 List the requirements for mutual exclusion.

5.8 What operations can be performed on a semaphore?

5.9 What is the difference between binary and general semaphores?

5.10 What is the difference between strong and weak semaphores?

5.11 What is a monitor?

5.12 What is the distinction between blocking and non-blocking with respect to messages?

5.13 What conditions are generally associated with the readers/writers problem?

6 Concurrency: Deadlock and Starvation

Reusable Resource Used safely by only one process at a time and is not depleted by that use (CPU, disk etc).

Consumable Resource A resource that can be created (produced) and destroyed (consumed). Examples are interrupts, signals and messages.

6.1 Deadlock

Existence of a deadlock:

- Mutual exclusion. Only one process may use a resource at a time. No process may access a resource unit that has been allocated to another process.
- Hold and wait. A process may hold allocated resources while awaiting assignment of other resources.
- No preemption. No resource can be forcibly removed from a process holding it.
- Circular wait. A closed chain of processes exists, such that each process holds at least one resource needed by the next process in the chain. An unresolvable circular wait is the definition of a deadlock.

If the first three conditions of the policy are present, there is a possibility of a deadlock. If all four are present, there is a deadlock. There are three approaches for dealing with deadlocks. One can prevent a deadlock by adapting a policy that eliminates one of the conditions. One can avoid a deadlock by making the appropriate dynamic choices based on the current state of resource allocation. One can attempt to detect the presence of a deadlock and take action to recover. A summary of theses approaches are shown in Figure 6.1.

6.2 Review questions

6.1 Give examples of reusable and consumable resources.

Approach	Resource Allocation Policy	Different Schemes	Major Advantages	Major Disadvantages
Prevention	Conservative; undercommits resources	Requesting all resources at once	• Works well for processes that perform a single burst of activity • No preemption necessary	• Inefficient • Delays process initiation • Future resource requirements must be known by processes
		Preemption	• Convenient when applied to resources whose state can be saved and restored easily	• Preempts more often than necessary
		Resource ordering	• Feasible to enforce via compile-time checks • Needs no run-time computation since problem is solved in system design	• Disallows incremental resource requests
Avoidance	Midway between that of detection and prevention	Manipulate to find at least one safe path	• No preemption necessary	• Future resource requirements must be known by OS • Processes can be blocked for long periods
Detection	Very liberal; requested resources are granted where possible	Invoke periodically to test for deadlock	• Never delays process initiation • Facilitates online handling	• Inherent preemption losses

Figure 6.1: Deadlock detection, prevention and avoidance approaches.

6.2 What are the three conditions that must be present for deadlock to be possible?

6.3 What are the four conditions that create deadlock?

6.4 How can the hold-and-wait condition be prevented?

6.5 List two ways in which the no-preemptive condition can be prevented.

6.6 How can the circular-wait condition be prevented?

6.7 What is the difference among deadlock avoidance, detection and prevention?

7 Memory Management

Memory management Subdividing memory dynamically to accommodate multiple processes.

Frame A fixed-length block of main memory.

Page A fixed-length block of data that resides in secondary memory. A page of data may temporarily be copied into a frame of main memory

Segment A variable-length block of data that resides in secondary memory. An entire segment may temporarily be copied into an available region of main memory (segmentation) or the segment may be divided into pages which can individually be copied into main memory (combined segmentation and paging).

Requirements that memory management intend to satisfy:

- **Relocation**: When swapping processes in and out of main memory, we might need to relocate the process to a different area of memory.
- **Protection**: Each process should be protected against unwanted interference by other processes.
- **Sharing**: Any protection mechanism must have the flexibility to allow several processes to access the same portion of main memory.
- **Logical Organization**: Programs organized into modules can realize a number of advantages (module references resolved at run time, degrees of protection on different modules, sharing modules among processes). Segmentation.
- **Physical Organization**: The task of moving information between main memory and secondary memory.

Technique	Description	Strengths	Weaknesses
Fixed Partitioning	Main memory is divided into a number of static partitions at system generation time. A process may be loaded into a partition of equal or greater size.	Simple to implement; little operating system overhead.	Inefficient use of memory due to internal fragmentation; maximum number of active processes is fixed.
Dynamic Partitioning	Partitions are created dynamically, so that each process is loaded into a partition of exactly the same size as that process.	No internal fragmentation; more efficient use of main memory.	Inefficient use of processor due to the need for compaction to counter external fragmentation.
Simple Paging	Main memory is divided into a number of equal-size frames. Each process is divided into a number of equal-size pages of the same length as frames. A process is loaded by loading all of its pages into available, not necessarily contiguous, frames.	No external fragmentation.	A small amount of internal fragmentation.
Simple Segmentation	Each process is divided into a number of segments. A process is loaded by loading all of its segments into dynamic partitions that need not be contiguous.	No internal fragmentation; improved memory utilization and reduced overhead compared to dynamic partitioning.	External fragmentation.
Virtual Memory Paging	As with simple paging, except that it is not necessary to load all of the pages of a process. Nonresident pages that are needed are brought in later automatically.	No external fragmentation; higher degree of multiprogramming; large virtual address space.	Overhead of complex memory management.
Virtual Memory Segmentation	As with simple segmentation, except that it is not necessary to load all of the segments of a process. Nonresident segments that are needed are brought in later automatically.	No internal fragmentation, higher degree of multiprogramming; large virtual address space; protection and sharing support.	Overhead of complex memory management.

Figure 7.1: Memory Management Techniques

7.1 Partitioning

7.1.1 Fixed partitioning

Fixed size partitions in memory, either equal size or not. Problems with fixed partitioning:

- Program too big to fit into a partition. Must use overlays to overcome.
- Internal fragmentation. Wasted space in a partition when a program is smaller than the size of the partition.

When swapping in equal-size partitions, it doesn't matter which partition is used since the size is equal. Which program to swap is decided by the scheduler. With unequal-size partitions, the process can be swapped into the smallest partition within which it will fit. This will create a separate queue on each partition, but it will minimize internal fragmentation. The other option is to employ a single queue for all processes. The smallest available partition will be selected for an incoming process. If all partitions are full, a swapping decision must be made.

7.1.2 Dynamic partitioning

Partitions of variable length and number. Figure 7.2 shows how it works. Problem: More and more external partitioning, as shown in Figure 7.2h.

Placement algorithms for dynamic partitioning in order of performance:

- First-fit: Scan memory from the beginning, choose first available.
- Next-fit: Scan memory from the location of the last placement, choose next available.
- Best-fit: Closest in size to the request. Leaves many small fragments.

Compaction: The OS iterates over the memory, moving all processes so that they are contiguous.

7.1.3 Buddy system

Think of a binary tree. When a request of memory enters, a leaf node will continue splitting into two equal size nodes (buddies) until it is as small as possible to hold the

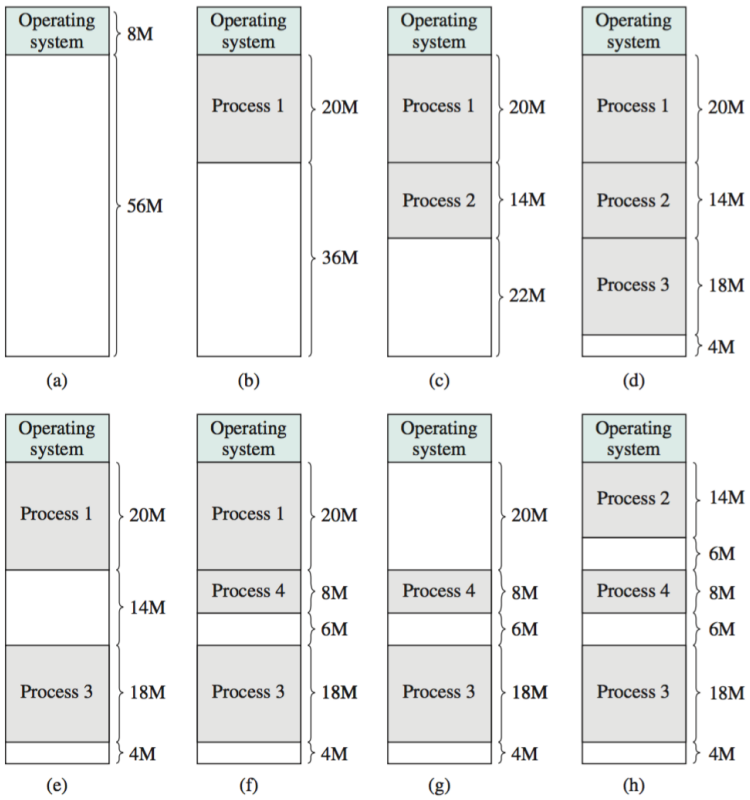


Figure 7.2: The effect of Dynamic Partitioning

requested memory. Whenever a pair of buddies become unallocated, they are coalesced into a single block. Figure 7.3 shows an example of a buddy system in action.

7.1.4 Relocation

When a process is swapped in and out of memory, it might be placed in different partitions. This presents a challenge with addresses in the process.

- A logical address is a reference to a memory location independent of the current assignment of data to memory; a translation must be made to a physical address before the memory acces can be achieved.
- A relative address is a type of logical address, expressed as a location relative to some know point.
- A physical address is a an actual location in main memory.

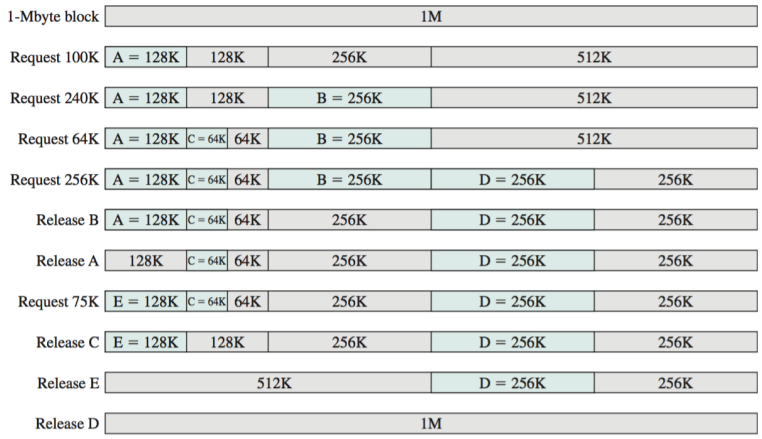


Figure 7.3: Example of Buddy System

Programs that employ relative addresses in memory are loaded using dynamic run-time loading.

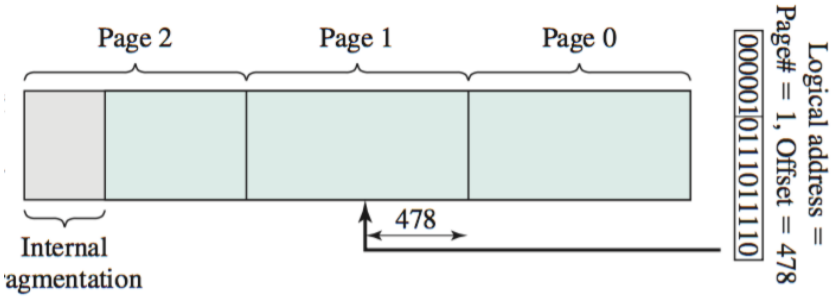
7.2 Paging

- Memory is partitioned into equal fixed-size chunks, relatively small.
- Processes is also divided into small fixed-size chunks of the same size.
- System maintains a page table for each process, so that the processes do not need to be contiguous.
- Very little internal fragmentation and no external fragmentation.

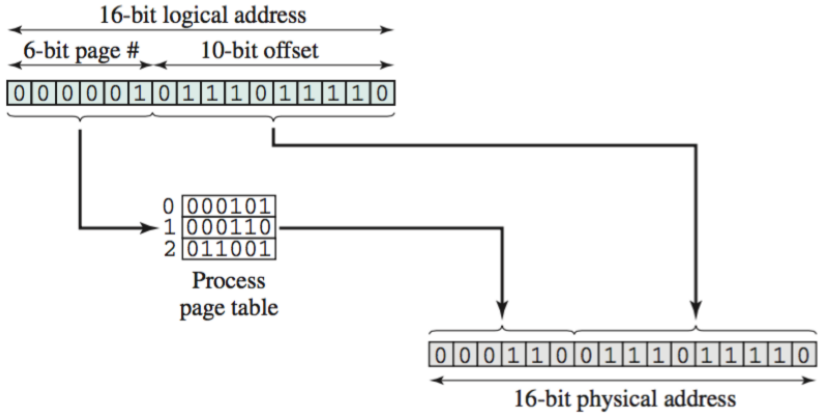
Figure 7.4 shows how a process is split into pages and how its logical address are translated to physical addresses using a page table. Figure 7.5 shows an example of paging during several swaps.

7.3 Segmentation

Similar to paging, but segments does not have to be equal in size (similar to dynamic partitioning). A logical address in segmentation consists of a segment number and an offset. Partitions in memory need not be contiguous. Segmentation suffers from external fragmentation, but since the process is broken up into a number of smaller



(a) Logical address



(b) Logigal-to-Physical address translation

Figure 7.4: Process is memory using pages and frames

pieces, the external fragmentation is less than for dynamic partitioning. Figure 7.6 shows an example of how segmentation works.

Characteristics of paging a segmentation led to a breakthrough in memory management; All the pages or all the segments of a process does not necessarily need to be in main memory during execution.

7.4 Review questions

7.1 What requirements is memory management intended to satisfy?

7.2 Why is the capability to relocate processes desirable?

7.3 Why is it not possible to enforce memory protection at compile time?

7.4 What are some reasons to allow two or more processes to all have access to a particular region of memory?

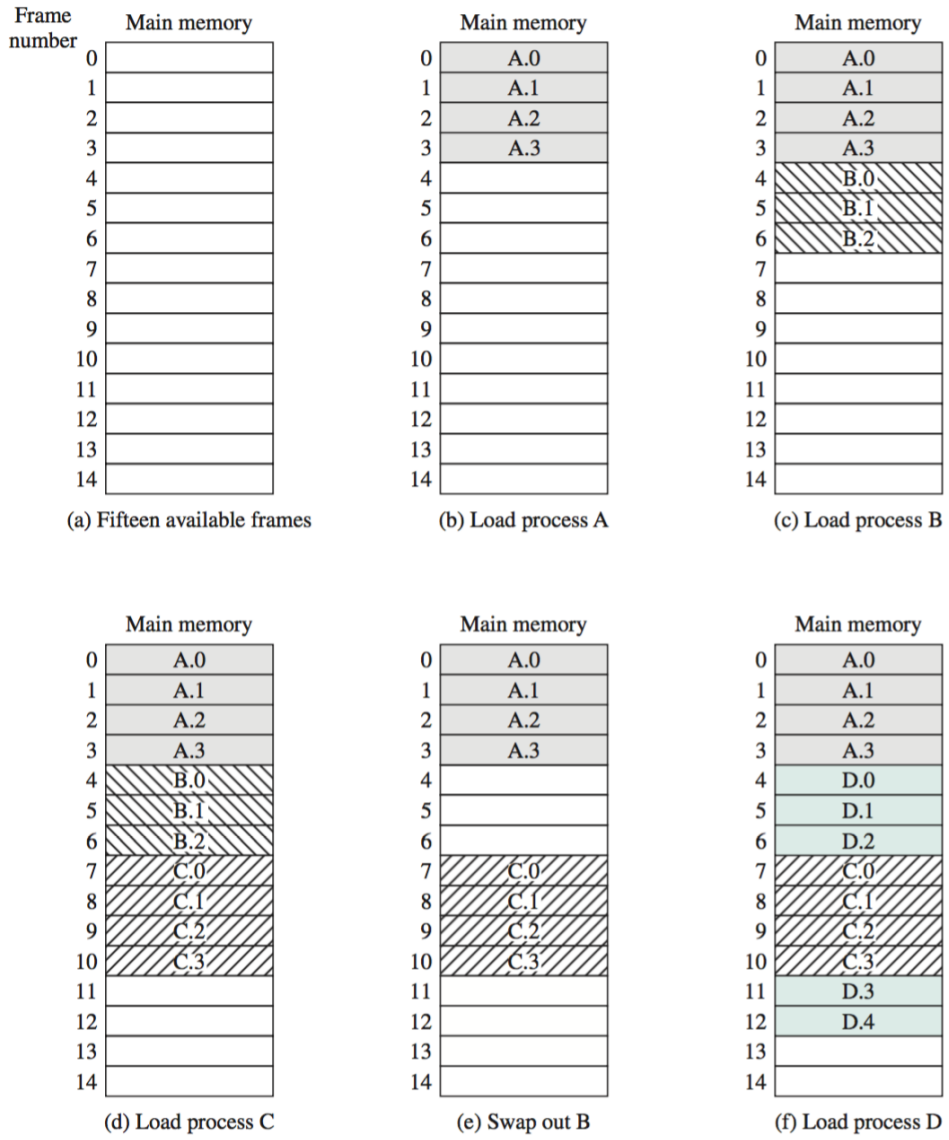
7.5 In a fixed-partition scheme, what are the advantages of using unequal-size partitions?

7.6 What are the difference between internal and external fragmentation?

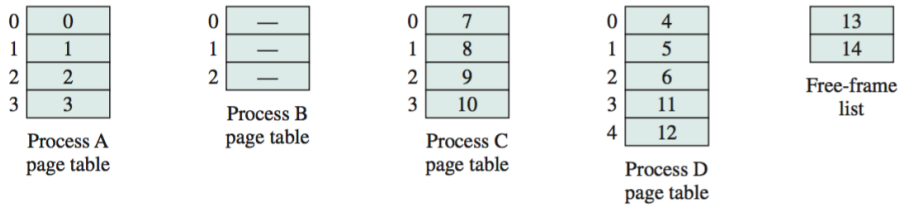
7.7 What are the distinctions among logical, relative and physical addresses?

7.8 What is the difference between a page and a frame?

7.9 What is the difference between a page and a segment?

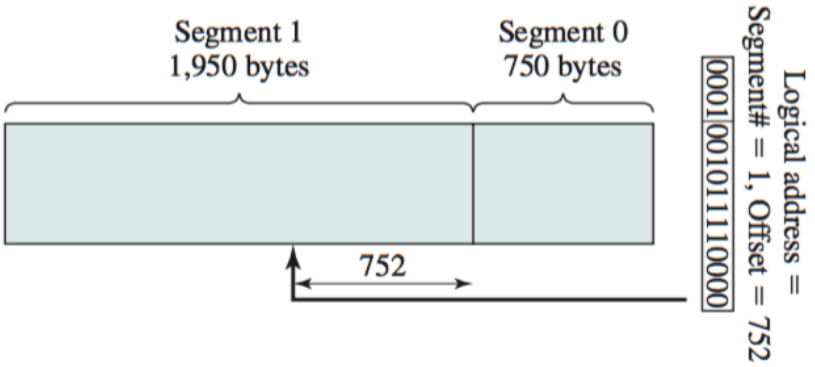


(a) Assignment of process to free frames

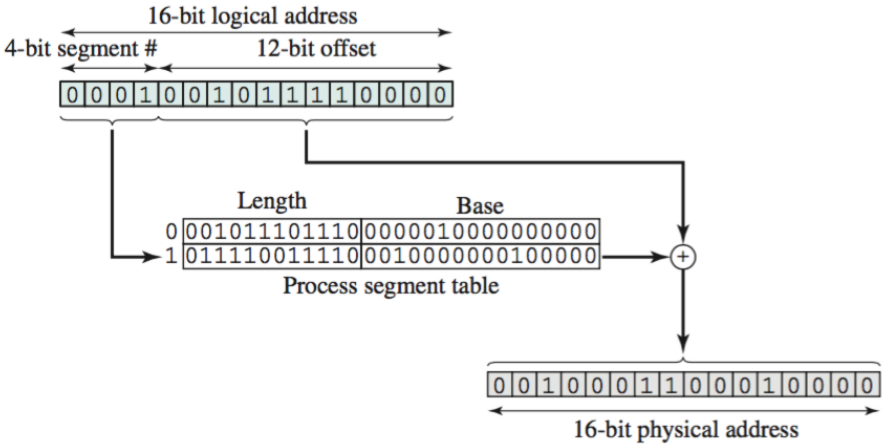


(b) Page tables for Figure 7.5a

Figure 7.5: Process is memory using pages and frames



(a) Logical address



(b) Logigal-to-Physical address translation

Figure 7.6: Segmented process is memory

8 Virtual Memory

Virtual memory A storage allocation scheme in which secondary memory can be addressed as though it were part of a main memory. The addresses a program may use to reference memory are distinguished from the addresses the memory system uses to identify physical storage sites, and program-generated addresses are translated automatically to the corresponding machine addresses. The size of the virtual storage is limited by the addressing scheme of the computer system and by the amount of secondary memory available and not by the actual number of main storage locations.

Virtual address The address assigned to a location in virtual memory to allow that location to be accessed as though it were part of main memory.

Virtual address space The virtual storage assigned to a process.

Address space The range of memory addresses available to a process.

Real address The address of a storage location in main memory.

Piece In this context, piece refers to either page or segment.

Resident set When a process is brought into memory, only some pieces, including the initial program and data piece is transferred to main memory. The resident set is the portion of a process that is actually in main memory at any time.

Thrashing The OS throws out a piece just before it used, and will have to go get that piece again almost immediately. Too much of this leads to a condition known as thrashing. Lot of research done on thrashing, today the OS tries to guess on which to throw out, based on recent history, which pieces are least likely to be used in the near future

TLB Translation lookaside buffer. A special high speed cache for page table entries that have been most recently used.

Frame locking A locked frame is a frame in main memory that may not be replaced when new pages are brought in. Much of the kernel of the OS as well as key control structures are held in locked frames.

LRU Last recently used, replacement policy.

FIFO First-in-first-out, replacement policy.

Working set The working set of information of a process at time t, is the collection of information referenced by the process during the process time interval. If the entire working set is in the resident set of a process, it is ready to execute.

With virtual memory, all address references are logical references that are translated at run time to real addresses. This allows a process to be located anywhere in main memory and for that location to change over time. Virtual memory also allows a process to be broken up into pieces. These pieces need not be contiguously located in main memory during execution and, indeed, it is not even necessary for all of the pieces of the process to be in main memory during execution.

Figure 8.1 summarizes characteristics of paging and segmentation with and without the use of virtual memory.

8.1 Hardware and control structures

For virtual memory to be practical and effective, two ingredients are needed.

- Hardware support for the paging and/or segmentation scheme to be employed.
- The OS must include software for managing the movement of pages and/or segments between secondary memory and main memory.

8.1.1 Paging

Figure 8.3 shows a typical implementation of the inverted page table approach. In this approach, the page number portion of the virtual address is mapped into a hash value, which is a pointer to the inverted page table.

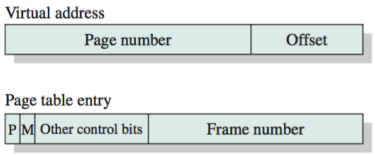
TLB[2] is used in most virtual memory schemes to overcome the problem of two physical memory accesses for fetching both the appropriate page table and then the desired data. An example of how the TLB works is displayed in Figure 8.4. The processor will first

[1] P = present bit (part of the resident set)
M = modified bit (if page not modified, it is not necessary to write it out upon replacement.
[2] TLB is explained in the beginning of the chapter.

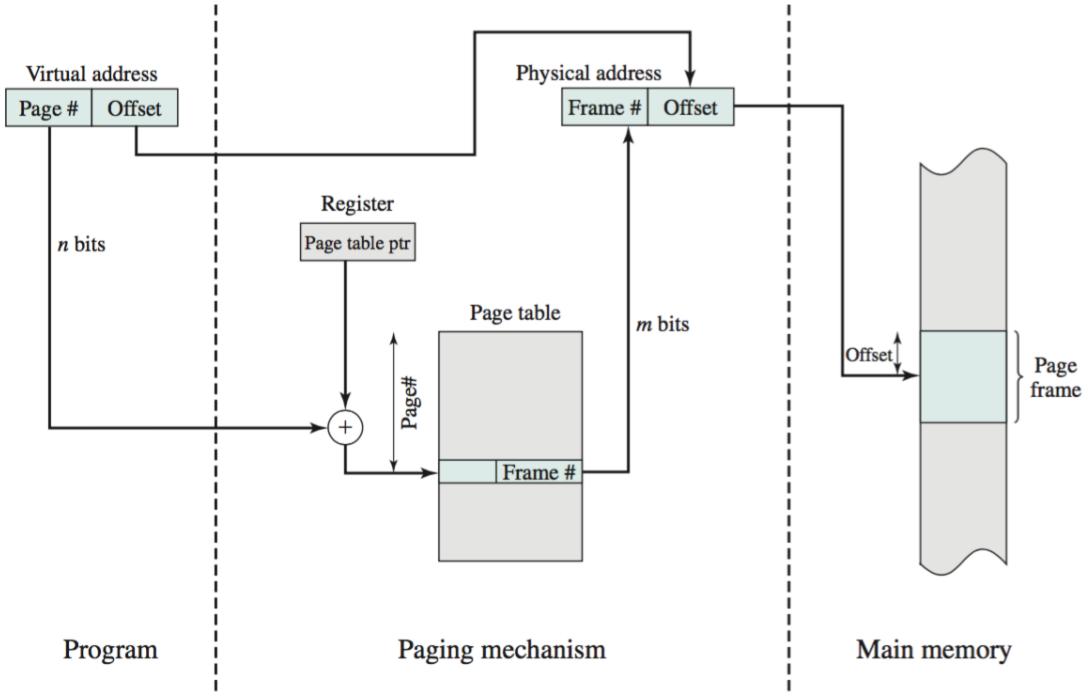
Simple Paging	Virtual Memory Paging	Simple Segmentation	Virtual Memory Segmentation
Main memory partitioned into small fixed-size chunks called frames.		Main memory not partitioned.	
Program broken into pages by the compiler or memory management system.		Program segments specified by the programmer to the compiler (i.e., the decision is made by the programmer).	
Internal fragmentation within frames.		No internal fragmentation.	
No external fragmentation.		External fragmentation.	
Operating system must maintain a page table for each process showing which frame each page occupies.		Operating system must maintain a segment table for each process showing the load address and length of each segment.	
Operating system must maintain a free-frame list.		Operating system must maintain a list of free holes in main memory.	
Processor uses page number, offset to calculate absolute address.		Processor uses segment number, offset to calculate absolute address.	
All the pages of a process must be in main memory for process to run, unless overlays are used.	Not all pages of a process need be in main memory frames for the process to run. Pages may be read in as needed.	All the segments of a process must be in main memory for process to run, unless overlays are used.	Not all segments of a process need be in main memory for the process to run. Segments may be read in as needed.
	Reading a page into main memory may require writing a page out to disk.		Reading a segment into main memory may require writing one or more segments out to disk.

Figure 8.1: Characteristics of paging and segmentation with virtual memory

examine the TLB and check if the desired page table is present. If not, the page is retrieved normally, and the TLB is updated with the new entry.



(a) Format of page table entry.[1]



(b) Address translation in a paging system

Figure 8.2: Address translation in a paging system

8.1.2 Segmentation

Both paging and segmentation have their strengths. Paging is transparent to the programmer and eliminates external fragmentation, and thus provides efficient use of main memory. Because the the pieces that are moved in and out of main memory are fixed size, it is possible to develop sophisticated memory management algorithms. Segmentation, which is visible to the programmer, has the ability to handle growing data structures, modularity, and support for sharing memory and protection. Some system are provided with hardware and OS software to provide both as seen in Figure 8.6. The user address space is broken up into a number of segments, and each segment is broken up into a number of fixed-size pages which are equal in length to a main memory frame.

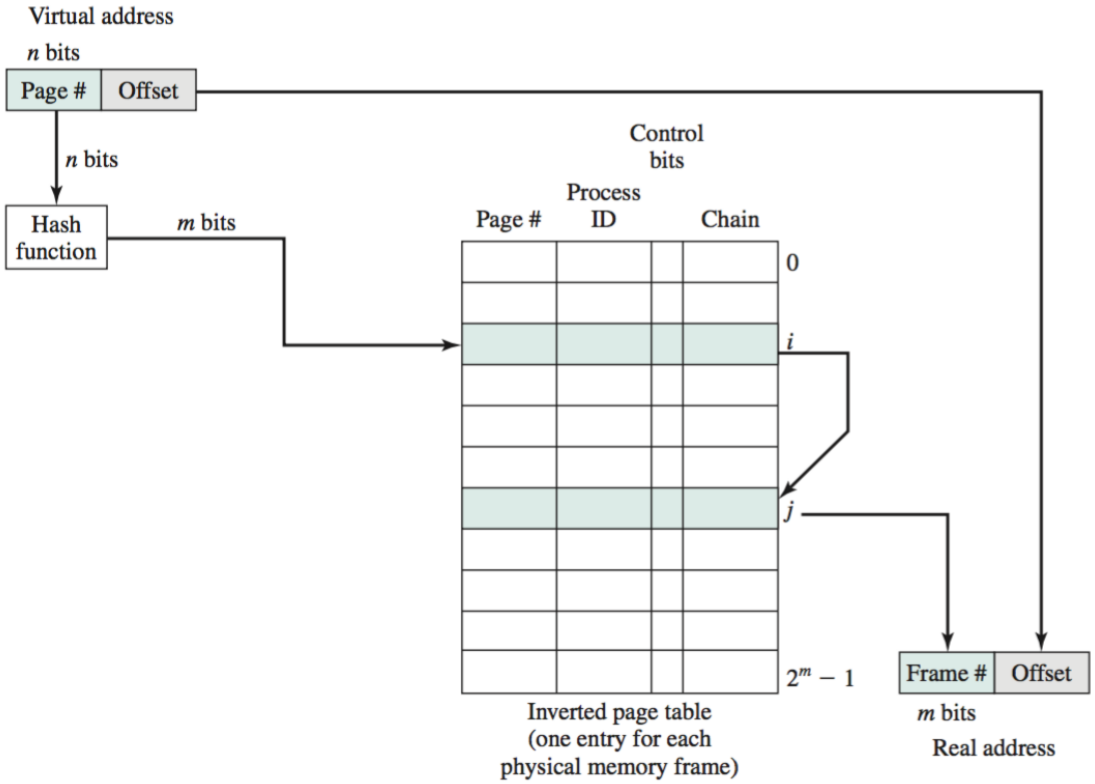


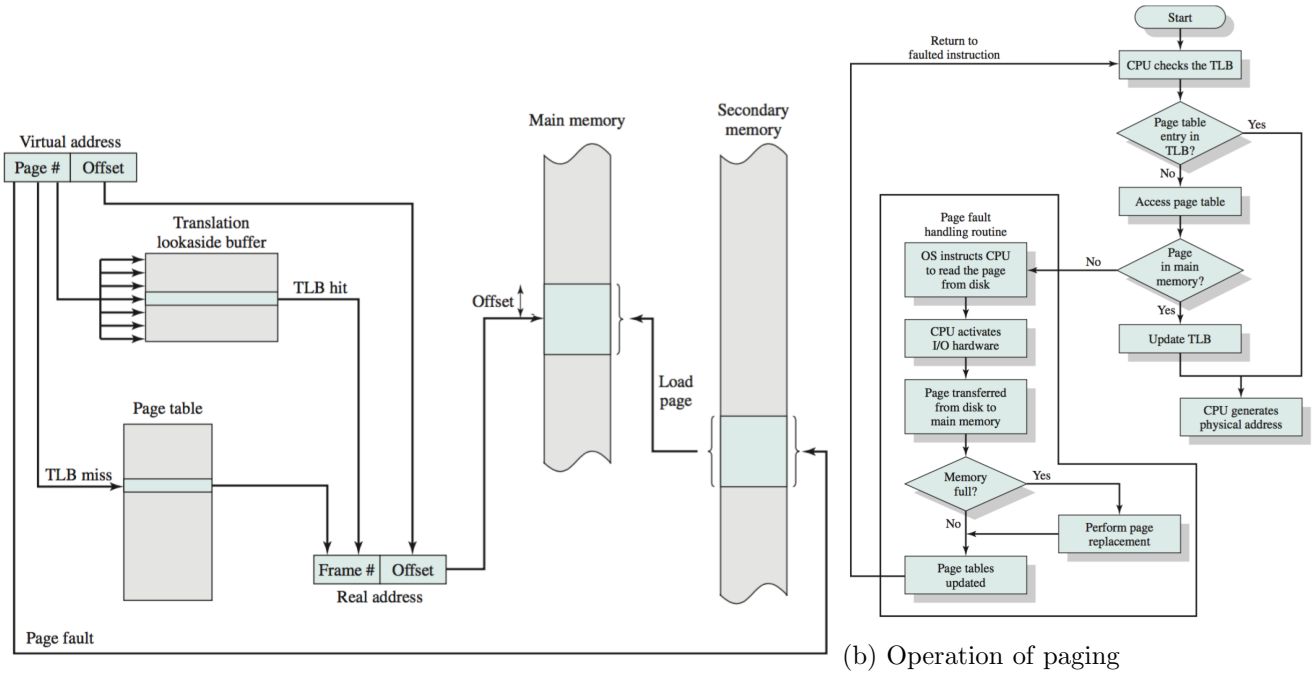
Figure 8.3: Inverted page table structure

8.2 Operating system software

The design of memory management in an OS depends on three fundamental areas of choice:

- Whether or not to use virtual memory techniques (harware dependent)
- The use of paging or segmentation or both (hardware dependent)
- The algorithms used for various aspects of memory management.

The choices related to the third point is the subject of this section. A number of design issues relate to OS support for memory management, and is presented in the following subsections.



(a) Example of how TLB is accessed

(b) Operation of paging and TLB

Figure 8.4: Translation Lookaside Buffer

8.2.1 Fetch policy

The fetch policy determines when a page should be brought into main memory. The two common alternatives:

- Demand paging: Brought into memory only when a reference is made to a location on that page.
- Prepaging: Pages other than the one demanded by a page fault are brought in. Bring in a number of contiguous pages stored in secondary memory.

8.2.2 Placement policy

The placement policy determines where in real memory a process piece is to reside. Best-fit, first-fit and next-fit for segmentation, usually irrelevant for paging.

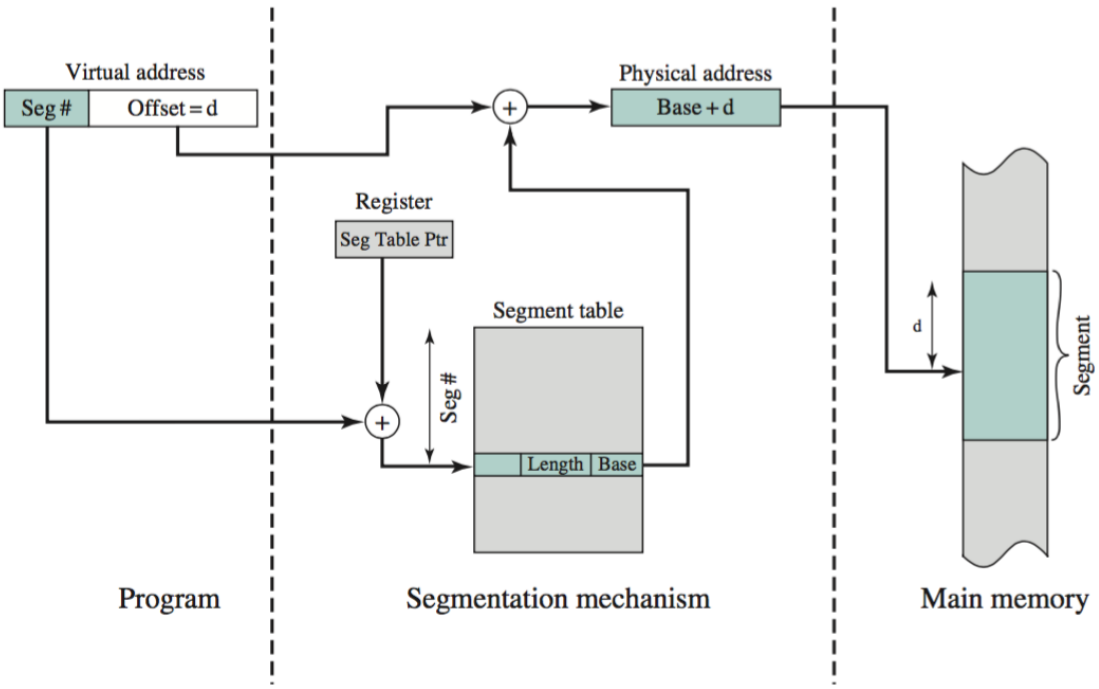


Figure 8.5: Address translation in a segmentation system

8.2.3 Replacement policy

The replacement policy determines which page in main memory to be replaced when a new page must be brought in.

- How many pages are to be allocated to each active process?
- Should the pages to replaced be limited to the process that caused the page fault or all the page frames in memory?
- Among the pages considered, which should be replaced?

Basic replacement algorithms:

- Optional: Selects the page for which the time to the next reference is the longest. Impossible to implement, since the OS cannot have perfect knowledge of future events. Use as a standard to judge other algorithms.
- LRU: Replaces the page in memory that has not been used for the longest time. Hard to implement.
- FIFO: Treats the page frames allocated to a process as a circular buffer, pages are

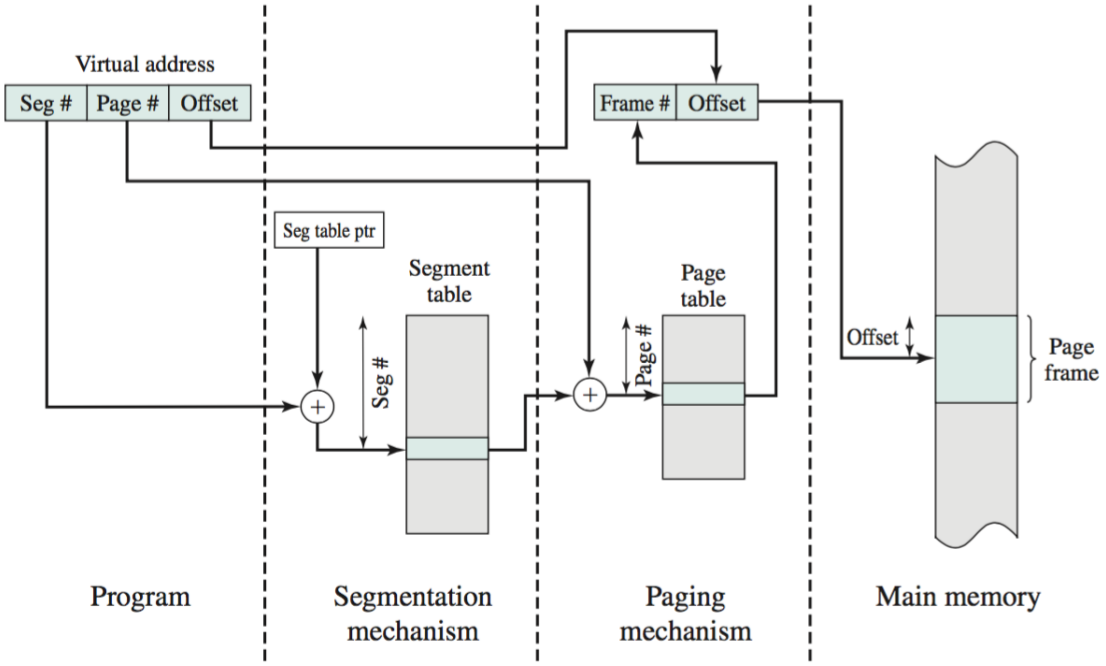


Figure 8.6: Address translation in a segmentation/paging system

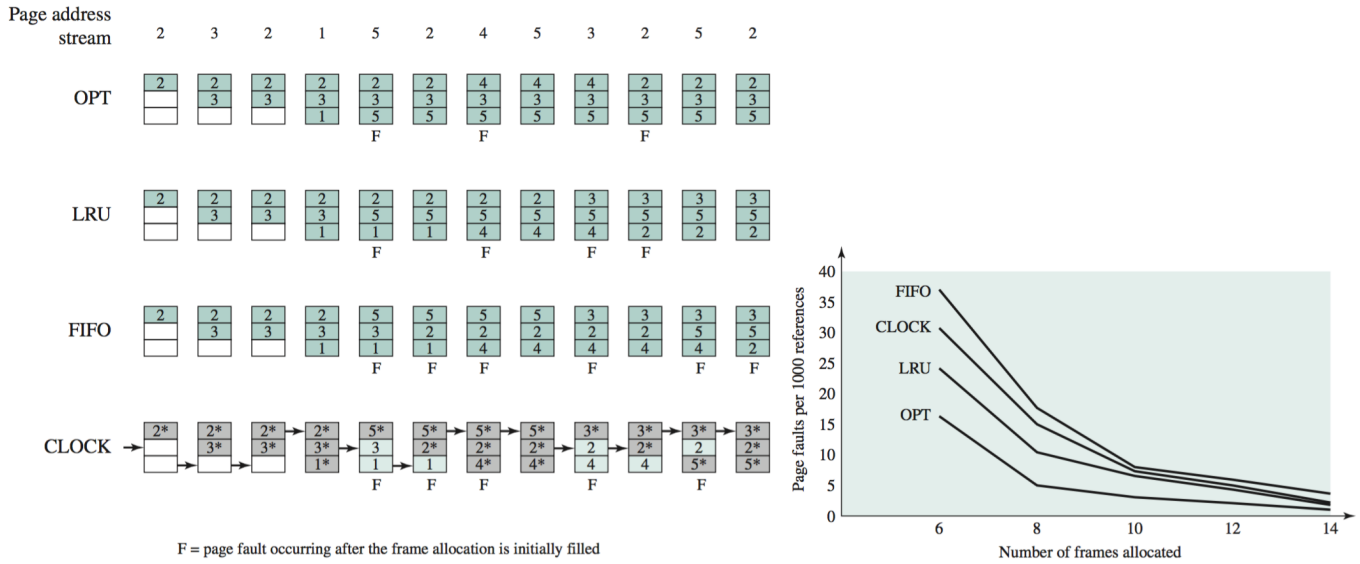
removed in a round-robin style. Simple to implement, but fails to identify pages that is used a lot throughout the execution of a program (2 and 5 in Figure 8.7a).

- Clock policy: Algorithms to approximate the performance of LRU while imposing little overhead. Simple clock policy: Similar to FIFO. Whenever a frame is used, a "use bit" is set to 1. When a page fault occurs, the OS scans through the circular buffer until it finds a frame with the use bit set to 0. When it passes a frame with the bit set to 1, it flips the bit to 0. The modified bit used in paging can also be included in this policy, to keep modified frames longer if possible.

8.2.4 Resident set management

When using paged virtual memory, the OS must decide how many pages to bring in (how much main memory to allocate) when a process starts.

- Less memory → more processes → Less swapping
- Small number of process pages in main memory → Higher page faults
- Beyond a certain size, more memory allocated to a process will have no effect on the



(a) Behaviour of optimal, LRU, FIFO and Clock replacement algorithms

(b) Comparison of replacement algorithms

Figure 8.7: Replacement policy algorithms

page fault rate (principle of locality)

A fixed-allocation police gives a process a fixed number of frames in maim memory. A variable-allocation policy allows the number of page frames allocated to a process to be varied over the lifetime of the process. A local replacement policy chooses only among the resident pages of the process that generated the page fault. A global replacement policy chooses between all unlocked pages in memory. Figure 8.8 shows combinations of resident set management.

	Local Replacement	**Global Replacement**
Fixed Allocation	• Number of frames allocated to a process is fixed. • Page to be replaced is chosen from among the frames allocated to that process.	• Not possible.
Variable Allocation	• The number of frames allocated to a process may be changed from time to time to maintain the working set of the process. • Page to be replaced is chosen from among the frames allocated to that process.	• Page to be replaced is chosen from all available frames in main memory; this causes the size of the resident set of processes to vary.

Figure 8.8: Resident set management

8.2.5 Cleaning policy

When should a modified page be written out to secondary memory? With demand cleaning, a page is written out when its has been selected for replacement. Precleaning writes batches of pages before their page frames are needed.

8.2.6 Load control

Determining number of processes that will be resident in main memory. Few processes → much time spent on swapping. Too many processes → frequent faulting will occur.

8.3 Review questions

8.1 What is the difference between simple paging and virtual memory paging?

8.2 Explain thrashing.

8.3 Why is the principle of locality crucial to the use of virtual memory?

8.4 What elements are typically found in a page table entry? Briefly define each element.

8.5 What is the purpose of a translation lookaside buffer?

8.6 Briefly define the alternative page fetch policies.

8.7 What is the difference between resident set management and page replacement policy?

8.8 What is the relationship between FIFO and clock page replacement algorithms?

8.9 What is accomplished by page buffering?

8.10 Why is it not possible to combine a global replacement policy and a fixed allocation policy?

8.11 What is the difference between a resident set and a working set?

8.12 What is the difference between demand cleaning and precleaning?

9 Uniprocessor Scheduling

Long-term scheduling The decision if to add a new process to the pool of processes to be executed. Thus controlling the degree of multiprogramming.

Medium-term scheduling The decision to add to the number of processes that are partially or fully in main memory. Part of the swapping function.

Short-term scheduling The decision as to which available process will be executed by the processor. Also called the dispatcher.

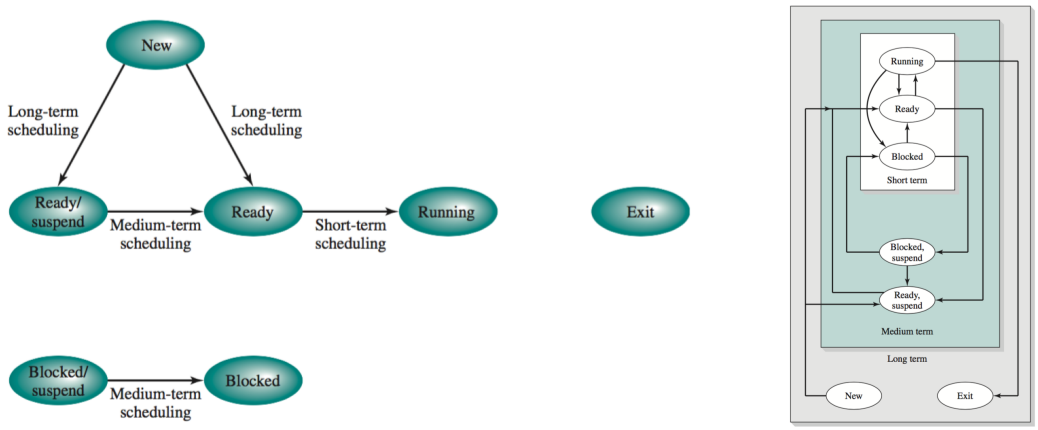


Figure 9.1: Scheduling and process state transitions

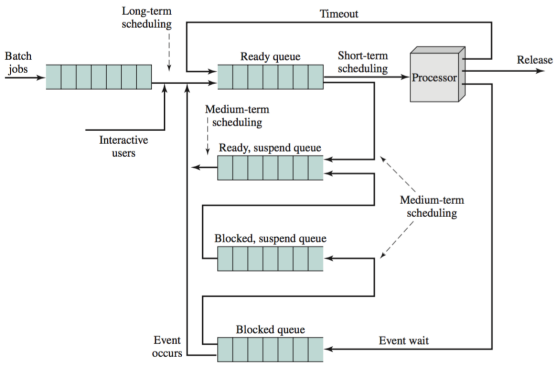


Figure 9.2: Queueing diagram for scheduling

I/O scheduling The decision as to which process' pending I/O request shall be handled by an available I/O device.

TAT Turnaround time. Time interval between the submission of a process and its completion. Time waiting + time served. Can be normalized by dividing on time served.

FCFS First-come-first-served. Strict queueing scheme implementing a FIFO.

Round robin Uses preempting based on a clock (time slice) and a queue of ready processes. Time slice short $\rightarrow$ short processes move through the system quickly $\rightarrow$ more overhead. Very long time slice = FCFS.

SPN Shortest process next. Risk: starvation of long processes. Not desirable in a time-sharing environment because of the lack of preemption.

SRT Shortest remaining time. Preemptive version of SPN. When a new process enters the ready queue, it is checked if its remaining time is shorter than the one running. If so, preempt.

HRRN Highest response ratio next. When the current process completes or is blocked, choose the process with the highest response ratio. $R = (w + s)/s$. Shorter jobs favored because of small denominator.

Feedback scheduling Focus on time spent in execution so far. Use of priority queues. Process starts with a high priority, and when it is preempted it is put the next lower priority queue. The execution time is doubled for each descending level to prevent starvation. Figure 9.3.

FSS Fair-share scheduler. Considers the execution history of a related group of processes, along with the individual execution history of each process in making scheduling decisions. Each fair-share group is provided with a virtual system that runs proportionally slower than a full system.

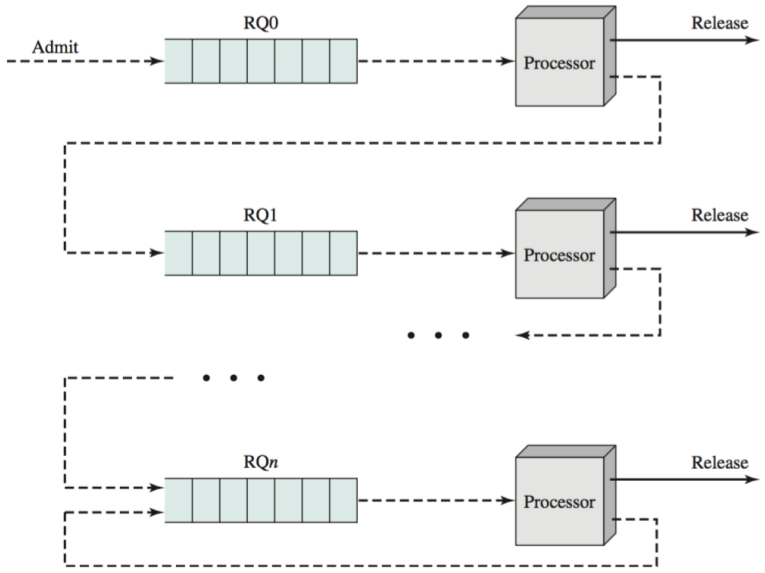


Figure 9.3: Multilevel feedback scheduling

	FCFS	Round Robin	SPN	SRT	HRRN	Feedback
Selection Function	max[w]	constant	min[s]	min[$s - e$]	max $\left(\frac{w+s}{s}\right)$	(see text)
Decision Mode	Non-preemptive	Preemptive (at time quantum)	Non-preemptive	Preemptive (at arrival)	Non-preemptive	Preemptive (at time quantum)
Throughput	Not emphasized	May be low if quantum is too small	High	High	High	Not emphasized
Response Time	May be high, especially if there is a large variance in process execution times	Provides good response time for short processes	Provides good response time for short processes	Provides good response time	Provides good response time	Not emphasized
Overhead	Minimum	Minimum	Can be high	Can be high	Can be high	Can be high
Effect on Processes	Penalizes short processes; penalizes I/O bound processes	Fair treatment	Penalizes long processes	Penalizes long processes	Good balance	May favor I/O bound processes
Starvation	No	No	Possible	Possible	No	Possible

Figure 9.4: Characteristics of various scheduling policies[a]

[a] w = time spent in system so far, waiting
e = time spent in execution so far
s total service time required by the process, including e

9.1 Review questions

9.1 Briefly describe the three types of processor scheduling.

9.2 What is usually the critical performance requirement in an interactive operating system?

9.3 What is the difference between turnaround time and response time?

9.4 For process scheduling, does a low-priority value represent a low priority or a high priority?

9.5 What is the difference between preemptive and nonpreemptive scheduling?

9.6 Briefly define FCFS scheduling.

9.7 Briefly define round-robin scheduling.

9.8 Briefly define shortest-process-next scheduling.

9.9 Briefly define shortest-remaining-time scheduling.

9.10 Briefly define highest-response-ratio-next scheduling.

9.11 Briefly define feedback scheduling.

10 Multiprocessor, Multicore and Real-Time Scheduling

Synchronization granularity Frequency of synchronization between the processes in a system.

Grain Size	Description	Synchronization Interval (Instructions)
Fine	Parallelism inherent in a single instruction stream	< 20
Medium	Parallel processing or multitasking within a single application	20–200
Coarse	Multiprocessing of concurrent processes in a multiprogramming environment	200–2,000
Very Coarse	Distributed processing across network nodes to form a single computing environment	2,000–1M
Independent	Multiple unrelated processes	Not applicable

Figure 10.1: Synchronization Granularity

10.1 Design issues

Assignment of processes to processors

I/O, main memory etc. specific processors? Otherwise, static or dynamic assigning? Static assignment is assigning a process permanently to one processor until its completion. Problem: queue for one processor can be empty, while another has a backlog. To prevent this, a common global queue can be used. Works well in a tightly coupled shared-memory architecture. Some means is needed to assign processes to processors. Two approaches have been used:

- Master/slave - key kernel functions of the OS always run on a particular processor. Master is responsible for scheduling jobs and delegate resources. Slave must request e.g. I/O from master. Conflict resolution simplified. Disadvantages: Failure of master brings down the system, and master can become a bottleneck.
- Peer - Kernel can execute on any processor, and a processor does self-scheduling from the pool of processes. Requires synchronizing of competing resource claims.

The use of multiprogramming on individual processors

Should a processor be multiprogrammed when a process is statically assigned? Depends on granularity, many processors available means that it is no longer paramount that every single processor are busy as much as possible.

Process dispatching

Complexities in scheduling algorithms may be unnecessary or even counterproductive in multiprocessors. The choice of specific scheduling discipline is much less important with multiple processors.

10.2 Scheduling

Thread scheduling

When threads of a process is running simultaneously on separate processors, we can exploit true parallelism. Approaches for thread scheduling:

- Load sharing - Global queue of all threads, processes not assigned to processor. Load distributed evenly, no centralized scheduler, different schemes can be used to select next thread. Three versions of load sharing: FSFC, Smallest numbers of threads first, Preemptive smallest number of threads first.
- Gang scheduling - Set of related threads is scheduled to run on a set of processors at the same time. Synchronized blocking reduced, less process switching, less scheduling overhead.
- Dedicated processor assignment - A process is allocated one processor per thread until it terminates. Total avoidance of process switching.
- Dynamic scheduling - Number of threads in a process can be altered during execution.

Two and two cores in a quadcore processor can share cache. In multicore thread scheduling, correlated threads should be placed in adjacent cores. The objective in multicore thread scheduling is to maximize the effectiveness of of the shared cache memory and therefore minimize the need for off-chip memory accesses.

10.3 Real-time scheduling

A real-time task is dependent on events in the outside world happening in real time. A hard real-time task is one that must meet its deadline, otherwise it will cause a fatal error. A soft real-time tasks has a deadline that is desirable but not mandatory.

Characteristics of real-time operating system requirements:

- Determinism - always respond to a request within a time limit
- Responsiveness - how long to service an interrupt
- User control - much broader in real-time than other
- Reliability
- Fail-soft operation - fail in such a way to preserve as much capability and data as possible

Classes of real-time scheduling algorithms:

- Static table-driven approaches - Static analysis performed, and schedule created
- Static priority-driven preemptive approaches - Static analysis assigning priorities, then a preemptive scheduler is used
- Dynamic planning-based approaches - Feasibility is determined at run time, resulting in a schedule for the new task
- Dynamic best effort approaches - No feasibility analysis, tries to meet all deadlines

Deadline scheduling performs scheduling based on a starting deadline or a completion deadline. Examples of deadline scheduling are fixed-priority scheduling, earliest deadline scheduling (with preemption), earliest deadline with unforced idle times, FCFS.

Rate monotonic scheduling

For RMS, the highest priority task is the one with the shortest period. It is possible to calculate if RMS will fail on a number of processes. Read about this in the book on page 457.

10.4 Review questions

10.1 List and briefly define five different categories of synchronization granularity.

10.2 List and define four techniques for thread scheduling.

10.3 List and briefly define three versions of load sharing.

10.4 What is the difference between hard and soft real-time tasks?

10.5 What is the difference between periodic and aperiodic real-time tasks?

10.6 List and briefly define five general areas of requirements for a real-time operating system.

10.7 List and briefly define four classes of real-time scheduling algorithms.

10.8 What items of information about a task might be useful in real-time scheduling?

11 I/O Management and Disk Scheduling

Block-oriented device Stores information in blocks, usually fixed size (disk, USB). Transfers one by one block.

Stream-oriented device Transfers data as a stream of bytes, no block structure (terminals, printers etc).

SSTF Shortest service time first. Select the disk I/O request that requires the least movement of the disk arm from its current position.

SCAN With SSTF, LIFO and PRI there is a possibility for starvation. SCAN (elevator algorithm) requires the arm to move in one direction only, satisfying all request en route until the last track. The service direction is then turned. Biased against recently traversed track, does not exploit locality as well as SSTF.

C-SCAN Circular SCAN. Only scanning in one direction, arm returned to the opposite end and scans again.

N-step-SCAN Split into sub queues of length N, and process one queue at a time. New requests are added to another queue.

F-SCAN FSCAN uses two queues. When one queue is processes, new requests are added to the other. Then switching roles.

RAID Redundant array of independent disks. Seven levels, zero through six.

- RAID is a set of physical disk drives viewed by the OS a a single logical drive.
- Data are distributed across the physical drives of an array in a scheme known as striping.
- Redundant disk capacity is used to store parity information, which guarantees data recoverability in case of a disk failure.

RAID 0 = Striping, RAID 1 = mirroring, RAID 3,4 = Parallel access, RAID 4,5,6 = Independent access.

External devices that engage in I/O can be grouped into three categories:

- Human readable - Communicating with the user (printers, terminals etc)
- Machine readable - Communicating with electronics (USB, sensors etc)
- Communication - Suitable for communicating with remote devices (modems, lines etc)

Key differences:

- Data (transfer) rate
- Application (software, drivers etc)
- Complexity of control
- Unit of transfer (stream of bytes, larger blocks)
- Data representation (encoding)
- Error conditions

Three I/O techniques:

- Programmed I/O (busy-wait)
- Interrupt-driven I/O
- Direct memory access (DMA)

Two objectives are paramount in designing the I/O facility:

- Efficiency - I/O often the bottleneck, especially disk I/O.
- Generality - Handle devices in a uniform manner.

I/O buffering can speed up I/O accesses, prevent blocking of processes, and allow processes to be swapped during I/O. Types of buffering:

- Single buffer
- Double buffer
- Circular buffer

Disk I/O transfer:

- Seek time (move disk arm to the required track) + rotational delay = access time (get in place for a read or write)
- Transfer time (time required for the transfer of data after access)

$$T = \frac{b}{rN}$$

where

T = transfer time

b = number of bytes to be transferred

N = number of bytes on a track

r = rotation speed, in revolutions per second

Thus, the total average access time can be expressed as

$$T_a = T_s + \frac{1}{2r} + \frac{b}{rN}$$

where T_s is the average seek time.

Figure 11.1: Transfer time

11.1 Disk scheduling policies

Random scheduling used as benchmarks to evaluate other techniques. Disk scheduling algorithms are summarized in Figure 11.2.

Name	Description	Remarks
Selection according to requestor		
Random	Random scheduling	For analysis and simulation
FIFO	First in first out	Fairest of them all
PRI	Priority by process	Control outside of disk queue management
LIFO	Last in first out	Maximize locality and resource utilization
Selection according to requested item		
SSTF	Shortest-service-time first	High utilization, small queues
SCAN	Back and forth over disk	Better service distribution
C-SCAN	One way with fast return	Lower service variability
N-step-SCAN	SCAN of *N* records at a time	Service guarantee
FSCAN	*N*-step-SCAN with *N* = queue size at beginning of SCAN cycle	Load sensitive

Figure 11.2: Disk scheduling algorithms

Disk cache

Disk cache is a buffer in main memory for disk sectors, exploiting the principle of locality.

Category	Level	Description	Disks Required	Data Availability	Large I/O Data Transfer Capacity	Small I/O Request Rate
Striping	0	Nonredundant	N	Lower than single disk	Very high	Very high for both read and write
Mirroring	1	Mirrored	$2N$	Higher than RAID 2, 3, 4, or 5; lower than RAID 6	Higher than single disk for read; similar to single disk for write	Up to twice that of a single disk for read; similar to single disk for write
Parallel access	2	Redundant via Hamming code	$N + m$	Much higher than single disk; comparable to RAID 3, 4, or 5	Highest of all listed alternatives	Approximately twice that of a single disk
	3	Bit-interleaved parity	$N + 1$	Much higher than single disk; comparable to RAID 2, 4, or 5	Highest of all listed alternatives	Approximately twice that of a single disk
Independent access	4	Block-interleaved parity	$N + 1$	Much higher than single disk; comparable to RAID 2, 3, or 5	Similar to RAID 0 for read; significantly lower than single disk for write	Similar to RAID 0 for read; significantly lower than single disk for write
	5	Block-interleaved distributed parity	$N + 1$	Much higher than single disk; comparable to RAID 2, 3, or 4	Similar to RAID 0 for read; lower than single disk for write	Similar to RAID 0 for read; generally lower than single disk for write
	6	Block-interleaved dual distributed parity	$N + 2$	Highest of all listed alternatives	Similar to RAID 0 for read; lower than RAID 5 for write	Similar to RAID 0 for read; significantly lower than RAID 5 for write

Figure 11.3: RAID levels

11.2 Review questions

11.1 List and briefly define three techniques for performing I/O.

11.2 What is the difference between logical I/O and device I/O?

11.3 What is the difference between block-oriented and stream-oriented devices? List examples.

11.4 Why would you expect improved performance using a double buffer rather than a single buffer for I/O?

11.5 What delay elements are involved in a disk read or write?

11.6 Briefly define the disk scheduling policies in Figure 11.2.

11.7 Briefly define the seven RAID levels.

11.8 What is the typical disk sector size? 512 bytes

12 File Management

Field Basic element of data. Characterized by length and data type (string, integer etc). E.g. the last name of a person.

Record A collection of related fields that can be treated as a unit by some program. E.g. an employee record.

File A collection of similar records. Treated as single entity by users, and may be referenced by name. Access control.

Database A collection of related data. Relationships among elements are explicit, designed for use by a number of different applications.

Block A block is the unit of I/O with secondary storage. Blocks can have fixed or variable length. Records must be organized as blocks when saved. Three methods of record blocking: fixed blocking (fixed length block holding numbers of records), variable-length spanned blocking (variable-length records in blocks with no unused space. Record may span two blocks), and variable-length unspanned blocking.

Portion Contiguous set of allocated blocks. Can range from a single block to the entire file.

FAT File allocation table. Used to keep track of the portions assigned to a file.

Disk allocation table Used to know which blocks on the disk are available using bit tables (vector, one bit per block), chained free portions (pointer and length value in each free portion), indexing (on variable size portions, one entry per free portion), free block list (list of free blocks on disk).

Volumes Collection of addressable sectors in secondary memory. A volume may be the result of assembling and merging smaller volumes, appears consecutive to the OS.

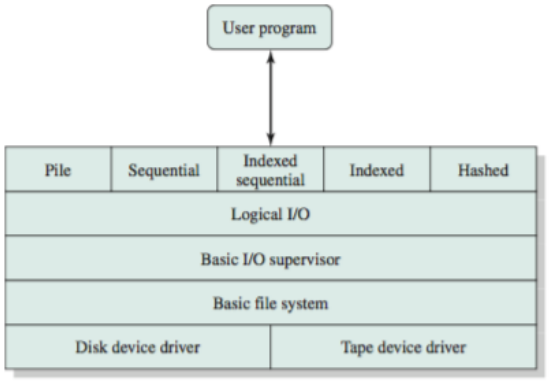


Figure 12.1: File system software architecture

The five blocks in the top layer of Figure 12.1 are five fundamental file organizations.

The pile

Data collected in the order they arrive. Each record consists of one burst of data. No structure, record access is by exhaustive search. Common for data not easy to organize.

The sequential file

Most common form of file structure. A fixed format is used for records. The first field in each record is called the key field. The key field uniquely identifies the record. Poor performance when reading/updating individual records.

The indexed sequential file

Records organized in sequence based on a key field. Two features added: an index to the file to support random access, and an overflow file. The index can be as simple as a sequential file with sorted key values to the section in file.

The indexed file

Limitations on previous two: Search for a record on the basis of some other attribute than the key field. The index file access records only through indexes. Two types of indexes: Exhaustive index (contains one entry for every record in the main file) and partial index (contains entries to records where the field of interest exists).

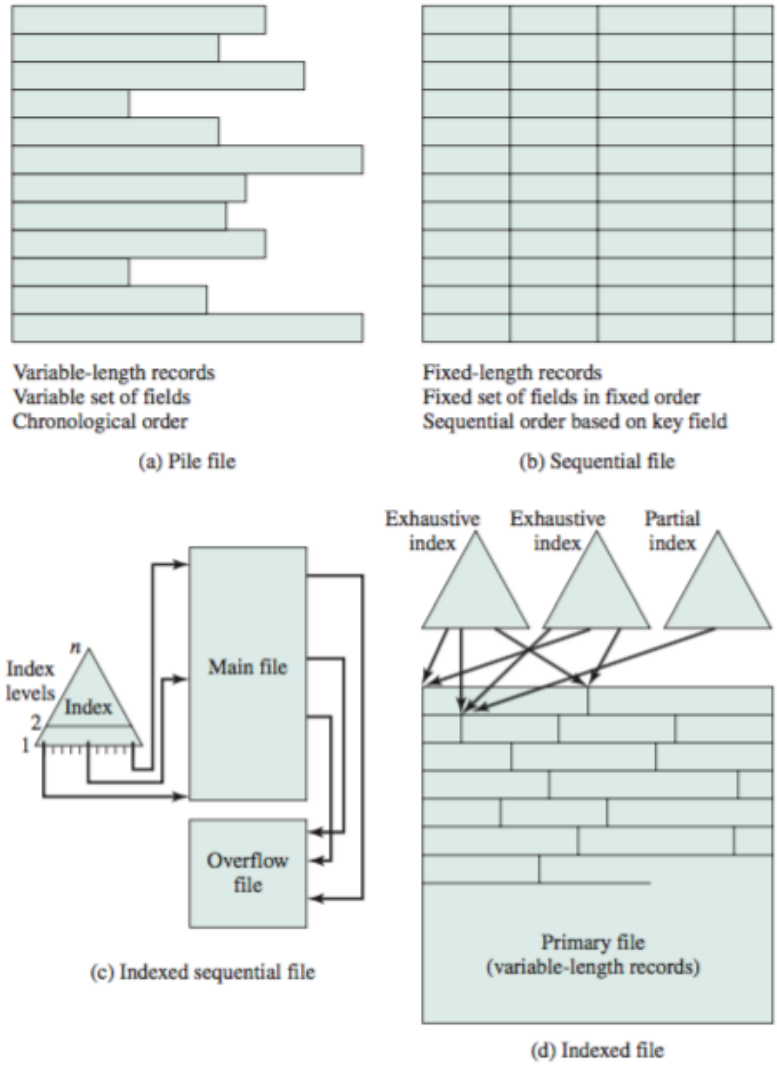


Figure 12.2: File system software architecture

The direct or hashed file

Exploits the capability found on disks to access directly any block of a known address. As with sequential and indexed sequential files, a key field is required in each record. The direct file makes use of hashing on the key value. Quick for accessing one record at a time.

12.1 File directories

Contains information about the files, including attributes, location and ownership. The directory is itself a file. Information about a file can be stored in its header, and the structure of a directory can be a sequential file of entries for each file. Inadequate when multiple users share a system, and you want to show info about file type etc. Operations that can be performed on a directory:

- Search
- Create file
- Delete file
- List directory
- Update directory

The universally adopted structure is the hierarchical tree-structure. Subdirectories may have other subdirectories and files.

	Contiguous	**Chained**	**Indexed**	
Preallocation?	Necessary	Possible	Possible	
Fixed or Variable Size Portions?	Variable	Fixed blocks	Fixed blocks	Variable
Portion Size	Large	Small	Small	Medium
Allocation Frequency	Once	Low to high	High	Low
Time to Allocate	Medium	Long	Short	Medium
File Allocation Table Size	One entry	One entry	Large	Medium

Figure 12.3: File allocation methods

File allocation methods:

- Contiguous allocation - allocated contiguous on disk, specifiying start block and length (number of blocks)
- Chained allocation - each block contains a pointer to the next block in the chain. Single entry to first block in allocation table. Dynamic allocation.
- Indexed allocation - File allocation table contains a separate one-level index for each file. Index has one entry per portion.

12.2 Review questions

12.1 What is the difference between a field and a record?

12.2 What is the difference between a file and a database?

12.3 What is a file management system?

12.4 What criteria are important in choosing file organization?

12.5 List and briefly define five file organizations.

12.6 Why is the average search time to find a record in a file less for an indexed sequential file than for a sequential file?

12.7 What are typical operations that may be performed on a directory?

12.8 What is the relationship between a pathname and a working directory?

12.9 What are typical access rights that may be granted or denied to a particular user for a particular file?

12.10 List and briefly define three blocking methods.

12.11 List and briefly define three file allocation methods.

13 Questions

- Differences between multiprocessing and multiprogramming?
- What is a batch operating system/batch environment?
- What is the difference between the functions of an operating system and an operating system kernel?
- Difference between nonprocess kernel, execution within user processes and process based operating system (page 139).
- Why can processes that did not put a lock on a semaphore, open the semaphore?
- Difference on putting a monitor on an object vs on a thread in Java?

13.1 Todo

- Repeat semaphores.
- Repeat producer/consumer example page 221-226.
- Maybe repeat working set page 373.
- Read intro of all Windows, Unix, Android sections.