Memory Management
*Frame - a fixed-length block of main memory
*Page - A fixed-length block of data that resides in secondary memory (such as disk). A page of data may temporarily be copies 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 devided into pages which can be individually copied into main memory (combined segmentation and paging).
*Memory Management Requirements (intended to satisfy the following requirements)
1)Relocation - Programmers typically do not know in advance which other programs will be resident in main memory at the time of execution of their program
- Active processes need to be able to be swapped in and out of main memory in order to maximize processor utilization
- Specifying that a process must be placed in the same memory region when it is swapped back in would be limiting.
- May need to relocate the process to a different area of memory.
2)Protection - Processes need to acquire permission to reference memory locations for reading or writing purposes
- Location of a program in main memory is unpredictable
- Memory references generated by a process must be checked at run time.
- Mechanisms that support relocation also support protection
3)Sharing - Advantageous to allow each process access to the same copy of the program rather than have their own seperate copy.
- Memory management must allow controlled access to shared areas of memory without compromising protection
- Mechanisms used to support relocation support sharing capabilities
4)Logical organization -Memory is organized as linear (like an array)
*Programs are written in modules
-modules can be written and compiled independently
-different degrees of protection given to modules (read-only, execute-only)
-sharing on a module level corresponds to the user's way of viewing the problem
-Segmentation is the tool that most readily satisfies requirements
5)Physical organization - Cannot leave the programmer with the responsibility to manage memory.
- Memory available for a program plus its data may be insufficient
*Overlaying allows various modules to be assigned the same region of memory but is time consuming to program
- Programmer does not know how much space will be available

Addressing Requirements

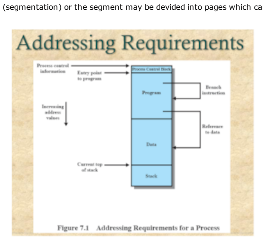


Figure 7.1 Addressing Requirements for a Process

*Memory Partitioning
-Memory management brings processes into main memory for execution by the processor
*involves virtual memory
*based on segmentation and paging
-Partitioning
*Used in several variations in some now-obsolete operating systems
*does not involve virtual memory

*Dynamic Partitioning - Partitions are created dynamically, so that each process is loaded into a partition of exactly the same size as the process.
Strengths: No internal fragmentation; more efficient use of main memory.
Weaknesses: Inefficient use of processor due to the need for compaction to counter external fragmentation.
*Partitions are of variable length and number
*Process is allocated exactly as much memory as it requires
*This technique was used by IBM's mainframe operating system, OS/MVT.
External Fragmentation:
*memory becomes more and more fragmented
*memory utilzation declines
Compaction:
*technique for overcoming external fragmentation
*OS shifts processes so that they are contiguous
*free memory is together in one block
*time consuming and wastes CPU time
*Simple Paging - Main memory is divided into a member of equal-size frames. Each process is divided into number of equal-size pages of the same length as frames. A process is loaded by loading all of its pages into available, not neccessarily contiguous frames.
Strengths: No external fragmentation.
Weaknesses: 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.
Strengths: No internal fragmentation; improved memory utilization and reduced overhead compared to dynamic partitioning.
Weaknesses: External fragmentation
*Virtual memory paging - As with simple paging, except that it is not neccessary to load all of the pages of a process. Nonresident segments that are needed are brought in later automatically.
Strengths: No internal fragmentation, higher degree of multiprogramming; large virtual address space; protection and sharing support.
Weaknesses: Overhead of complex memory management.
*Placement Algorithms
-Best Fit - chooses the block that is closest in size to the request.
-First-fit - begins to scan memory from the beginning and chooses the first available block that is large enough
-Next-fit - begins to scan memory from the location of the last placement and chooses the next available block that is large enough

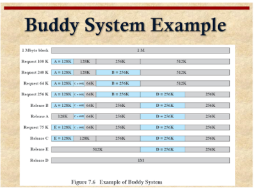
*Addresses
-Logical - reference to a memory location independent of the current assignment of data to memory.
-Relative - address is expressed as a location relative to some known point.
-Physical or Absolute - actual location in main memory.
*Paging
-Partition memory into equal fixed-size chunks that are relatively small.
-Process is also divided into small fixed-size chunks of the same size.
*Pages: chunks of a process
*Frames: available chunks of memory
*Page Table
-Maintained by operating system for each process.
-Contains the frame location for each page in the process
-Processor must know how to access for the current process
-Used by processor to produce a physical address
*Logical Addresses
Relative adress = 1502 which is the start of the first process at adress 0.
offset: number of positions that you can jump within a page (last 10 bits).
offset range: 0000000000-1111111111
page size = 1k, log2(1024) = 10. The offset = 10 bits(least significant)
page numer: first 6 bits in the 16 bit address.
base register = page number + offset filled with zeros
*To get the the location that you want(physical address), use the page number + offset
physical address = 6-bit page number + 10-bit offset
*Segmentation
- A program can be subdivded into segments
*may vary in length
*there is a maximum length
-Addressig consists of two parts:
1)segment number
2)an offset
-Similar to dynamic partitioning
-Eliminates internal fragmentation
*Security Issues
-If a process declares that a portion of memory may be shared by other designated processes then the security service of the OS must ensure that only the designated processes have access
-If a process has not declared a portion of its memory to be sharable, then no other process should have access to the contents of that portion of memory.
*Buffer Overflow Attacks
-Security threat related to memory management
-Also known as a buffer overrun
-Can occur when a process attempts to store data beyond the limits of a fixed-sized buffer.
-One of the most prevalent and dangerous types of security attacks
*Memory Management
-One of the most important and complex tasks of an operating system.
-needs to be treated as a resource to be allocated to and shared among a number of active processes
-desireable to maintain as many processes in main memory as possible
-desirable to free programmers from size restriction in program development
-basic tools are paging and segmentation(possible to combine)
*paging - small fixed-sized blocks
*segmentation - pieces of varying size
*Virtual Memory - A storage allocation scheme in which secondary memory can be addressed as though it were part of 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 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.
*Hardware and Control Structures
-Two characteristics fundamental to memory management
1) all memory references are logical addresses that are dynamically translated into physical addresses at run time.
2)a process may be broken up into a number of pieces that don't need to be contiguously located in main memory dur. execution
-If these two characteristics are present, it is not necessary that all of the pages or segments of a process be in main memory dur. execution
*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 mem.
*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.
*Execution of a Process
-Operating system brings into main memory a few pieces of the program
-Resident set - portion of process that is in main memory.
-An interrupt is generated when an address is needed that is not in main memory
-Operating system places the process in a blocking state
-Piece of process that contains the logical address is brought into main memory
*operating system issues a disk i/o read request
*another process is dispatched to run while the disk i/o takes place
*an interrupt is issued when disk i/o is complete, which causes the operating system to place the affected process in the Ready state.
Inverted Page Table
-Page number portion of a virtual address is mapped into a hash value
-hash value points to inverted page table
-Fixed proportion of real memory is required for the tables regardless of the number of processes or virtual pages supported.
-Structure is called inverted because it indexes page table entries by frame number rather than by virtual page number.
*Each entry in the page table inclues:
-Page number
-Process identifier - the process that onws this page
-Control bits - includes flags and protection and locking information
-Chain pointer - the index value of the next entry in the chain
*Translation Lookaside Buffer (TLB)
-Each virtual memory reference can cause two physical memory accesses:
*one to fetch the page table entry
*one to fetch the data
-To overcome the effect of double the memory access time, most virtual memory schemes make use of a special high-speed cache called a translation lookaside buffer
*Associative Mapping
-The TLB only contains some of the page table entries so we cannot simply index into the TLB based on page number
-Each TLB entry must include the page number as well as the complete page table entry
-The processor is equipped with hardware that allows it to interrogate simultaneously a number of TLB entries to determine if there is a match on page number
*Page Size
-The smaller the page size, the lesser the amount of internal fragmentation
-however, more pages are required per process
-more pages per process means larger page tables
-for large programs in a heavily multiprogrammed environment some portion of the page tables of active processes must be in virtual memory instead of main memory
-the physical characteristics of most secondary-memory devices favor a larger page size for more efficient block transfer of data
*Contemporary programming techniques used in large programs tend to decrease the locality of references within a process.
1.The design issue of page size is related to the size of physical main memory and program size
2.Main memory is getting larger and address space used by applications is also growing
3.most obvious on personal computers where applications are becoming increasingly complex

**Operating System Software
-The design of the memory management portion of an operating system depends on three fundamental areas of choice:
1)whether or not to use virtual memory techniques
2)the use of paging or segmentation or both
3)the algorithms employed for various aspects of memory management
Fetch Policy
-Determines when a page should be brought into memory
1)Demand paging
*only brings pages into main memory when a reference is made to a location on the page
*many page faults when process is first started
*principle of locality suggests that as more and more pages are brought in, most future references will be to pages that have recently been brought in, and page faults should drop to a very low level.
2)Prepaging
*pages other than the one demanded by a page fault are brought in
*exploits the characteristics of most secondary memory devices
*if pages of a process are stored contiguously in secondary memory it is more efficient to bring in a number of pages at one time
*ineffective if extra pages are not referenced
*should not be confused with "swapping"

*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.
Strengths: Simple to implement; little operating system overhead.
Weaknesses: Inefficient use of memory due to internal fragmentation; maximum number of active processes is fixed.
* equal-size partitions
* any process whose size is less than or equal to the partition size can be loaded into an available partition.
* The operating system can swap out a process if all partitions are full and no process is in the Ready or Running state.
Disadvantages:
-A program may be too big to fit in a partition
*program needs to be desgned with the use of overlays.
-Main memory utilization is inefficient
*Any program, regardless of size, occupies an entire partition
***Internal fragmentation**
*wasted space due to the block of data loaded being smaller than the partition
-The number of partitions specified at system generation time limits the number of active processes in the system.
-Small jobs will not utilize partition space efficiently.
Unequal Size Partitions - using unequal size partitions helps lessen the problems
-programs up to 16M can be accommodated without overlays
-Partitions smaller than 8M allow smaller programs to be accommodated with less internal fragmentation

*Buddy System
-Comprised of fixed and dynamic partitioning schemes.
-Space available for allocation is treated as a single block.
-Memory blocks are available of size 2^k words, L<=K<=U, where
* 2^L = smallest size block that is allocated
* 2^U = largest size block that is allocated; generally 2^U is the size of the entire memory available for allocation.

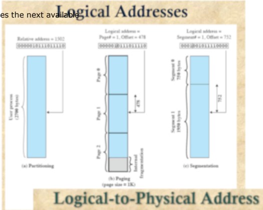
Buddy System Example



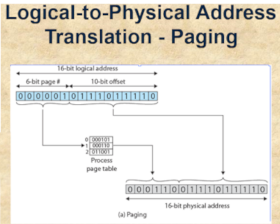
1 Mbyte block	1 M				
Request 100 K	A = 128K	128K	256K	512K	
Request 240 K	A = 128K	128K	B = 256K	512K	
Request 64 K	A = 128K	C = 64K 64K	B = 256K	512K	
Request 256 K	A = 128K	C = 64K 64K	B = 256K	D = 256K	256K
Release B	A = 128K	C = 64K 64K	256K	D = 256K	256K
Release A	128K	C = 64K 64K	256K	D = 256K	256K
Request 75 K	E = 128K	C = 64K 64K	256K	D = 256K	256K
Release C	E = 128K	128K	256K	D = 256K	256K
Release E	512K			D = 256K	256K
Release D	1M				

Figure 7.6 Example of Buddy System

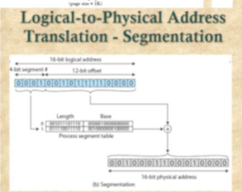
Logical Addresses



Logical-to-Physical Address Translation - Paging



Logical-to-Physical Address Translation - Segmentation



*Implications
-More processes may be maintained in main memory
*only load in some of the pieces of each process
*with so many processes in main memory, it is very likely a process wil be in the ready state at any particular time
-A process may be larger than all of main memory.
*Real memory - main memory, the actual RAM
*Virtual memory - memory on disk, allows for effective multiprogramming and relieves the user of tight constraints of main memory.
*Thrashing - A state in which the system spends most of its time swapping process pieces rather than executing instructions.
-to avoid this, the operating system tries to guess, based on recent history, which pieces are least likely to be used in the near future.
*Principle of locality
-Program and data references within a process tend to cluster
-Only a few pieces of a process will be needed over a short period of time
-Therefore it is possible to make intelligent guesses about which pieces will be needed in the future
-Avoids thrashing
*Paging Behavior
-During the lifetime of the process, references are confined to a subset of pages.
*Support Needed for Virtual Memory
-For virtual memory to be practical and effective:
*hardware must support paging and segmentation
*operating system must include software for managing the movement of pages and/or segments between secondary memory and main memory
*Paging
-The term virtual memory is usually associated with systems that employ paging
-Use of paging to achieve virtual memory was first reported for the Atlas computer
-Each process has its own page table
*each page table entry contains the frame number of the corresponding page in main memory
*Memory management Formats
(Paging)
-Virtual Address :
*Page Number
*Offset - tells you the size of the page
-Page Table Entry
*Present bit - if it is in virutal memory or main memory
*Modified bit
*Other Control BIts
*Frame Number - information to create the physical address
(segmentation)
-Virtual Address
*Segment Number
*Offset
-Segment Table Entry
*Present bit
*Modified bit - if the segment was modified when uploaded to main memory
*Other Control Bits
*Length
*Segment Base

Combined Segmentation and paging
-virtual addres:
*Segment number
*Page number
*Offset
-Segment Table Entry:
*Control bits
*Length
*Segment Base
-Page Table Entry:
*Present bit
*Modified bit
*Other Control Bits
*Frame Number

*Segmentation
-Segmentation allows the programmer to view memory as consisting of multiple address spaces or segments
Advantages:
*simplifies handling of growing data structures
*allows porgrams to be altered and recompiled independently
*lends itself to sharing data among processes
*lends itself to protection
*Segment Organization
-Each segment table entry contains the starting address of the corresponding segment in main memory and the length of the segment
-A bit is needed to determine if the segment is already in main memory
-Another bit is needed to determine if the segment has been modified since it was loaded in main memory
*Combined Paging and Segmentation
-In a combined paging/segmentation system a user's address space is broken up into a number of segments. Each segment is broken up into a number of fixed-sized pages which are equal in length to a main memory frame..
*segmentation is visible to the programmer
*Paging is transparent to the programmer
*Proctection and Sharing
-Segmentation lends itself to the implementation of protection and sharing policies
-Each entry has a base address and length so inadvertent memory access can be controlled
-Sharing can be achieved by segments referencing multiple processes

Free Block List

There are two effective techniques for storing a small part of the free block list in main memory:
-the list can be treated as a push-down stack with the first few thousand elements of the stack kept in main memory
-the list can be treated as a push-down stack with the first few thousand elements of the stack kept in main memory

Depending on the size of the disk, either 24 or 32 bits will be needed to store a single block number
-the size of the free block list is 24 or 32 times the size of the corresponding bit table and must be stored on disk

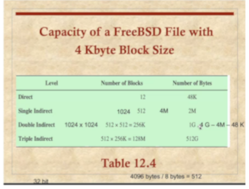
The list of the numbers of all free blocks is maintained in a reserved portion of the disk
-Each block is assigned a number sequentially

Volumes
-A collection of addressable sectors in secondary memory that an OS or application can use for data storage
-The sectors in a volume need not be consecutive on a physical storage device they need only appear that way to the OS or application
-A volume may be the result of assembling and merging smaller volumes

*Operating System Software
-The design of the memory management portion of an operating system depends on three fundamental areas of choice:
1)whether or not to use virtual memory techniques
2)the use of paging or segmentation or both
3)the algorithms employed for various aspects of memory management
Fetch Policy
-Determines when a page should be brought into memory
1)Demand paging
*only brings pages into main memory when a reference is made to a location on the page
*many page faults when process is first started
*principle of locality suggests that as more and more pages are brought in, most future references will be to pages that have recently been brought in, and page faults should drop to a very low level.
2)Prepaging
*pages other than the one demanded by a page fault are brought in
*exploits the characteristics of most secondary memory devices
*if pages of a process are stored contiguously in secondary memory it is more efficient to bring in a number of pages at one time
*ineffective if extra pages are not referenced
*should not be confused with "swapping"
*Placement Policy
-Determines where in real memory a process piece is to reside
-Important design issue in a segmentation system
-Paging or combined paging with segmentation placing is irrelevant because hardware performs functions with equal efficiency
-for NUMA systems an automatic placement strategy is desirable
*Replacement Policy
-Deals with the selection of a page in main memory to be replaced when a new page must be brought in
*objective is that the page that is removed be the page least likely to be referenced in the near future
-The more elaborate the replacement policy the greater the hardware and software overhead to implement it
*Frame Locking
-When a frame is locked the page currently stored in that frame may not be replaced
*kernel of the OS as well as key control structures are held in locked frames
*I/O buffers and time-critical areas may be locked into main memory frames
*locking is achieved by associating a lock bit with each frame

**Basic Algorithms
-Optimal
-Least recently used (LRU)
-Firstin-first-out
-Clock
**Optimal Policy
-Selects the page for which the time to the next reference is the longest
-Produces three page faults after the frame allocation has been filled
~cannot be implemented

Capacity of a FreeBSD File with 4 Kbyte Block Size



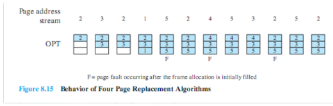
Level	Number of Blocks	Number of Bytes
Direct	12	48K
Single Indirect	1024 512	4M 2M
Double Indirect	1024 x 1024 512 × 512 = 256K	1G 4 G − 4M − 48 K
Triple Indirect	512 × 256K = 128M	512G

Table 12.4
4096 bytes / 8 bytes = 512
32 bit

File System Calculation
8 bits = 1 byte
Direct = 12 blocks, 48K bytes
Single Indirect = Block Size/Address-Size (bytes)
Double Indirect = # Blocks Single Indirect * # Blocks Single Indirect
Triple Indirect = #Blocks Single * (#Blocks Single)/2
Number of bytes = #Blocks * Block Size (make sure you note whether you get K or G, K = Num of Blocks / 1024, G if you go beyond K)

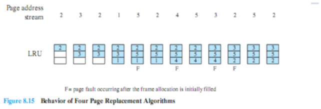
i.e. 4Kbyte with 32-bit address
Direct = 12, 48K
Single = 4096/(32/8) = 1024 = 1K
Num of bytes = 1K * 4Kbyte Block size = 4M
etc.
In reality, subtract 4G -4M - 48K since limit is 4G with 32 bits

~EXAM QUESTION OPTIMAL POLICY ~


Page address stream
2 3 2 1 5 2 4 5 3 2 5 2
OPT
F
F
F
F= page fault occurring after the frame allocation is initially filled
Figure 8.15 Behavior of Four Page Replacement Algorithms

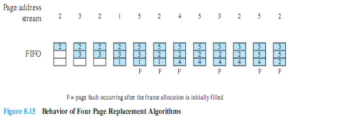
Fault #1 (F): 5 replaces 1 because 1 in not used again in the future.
Fault #2 (F): 4 replaces 2 because process 2 comes much later in the future than the others
Fault #3 (F): 2 replaces 4 because 4 is not used again the the future.

*Least recently used (LRU)
-Replaces the page that has not been referenced for the longest time
-By the principle of locality, this should be the page least likely to be referenced in tl
-Difficult to implement
*one approach is to tag each page with the time of last reference
*this requires a great deal of overhead


Page address stream
2 3 2 1 5 2 4 5 3 2 5 2
LRU
F
F
F
F
F= page fault occurring after the frame allocation is initially filled
Figure 8.15 Behavior of Four Page Replacement Algorithms

Fault #1 (F): 5 replaces 3 because 3 is the oldest time stamp
continue until completion...

*First-in-First-out(FIFO)
-Treats page frames allocated to a process as a circular buffer
-pages are removed in round-robin style
*simple replacement policy to implement
-page that has been in memory the longest is replaced


Page address stream
2 3 2 1 5 2 4 5 3 2 5 2
FIFO
F
F
F
F
F
F
F= page fault occurring after the frame allocation is initially filled
Figure 8.15 Behavior of Four Page Replacement Algorithms

Fault #1 (F): 2 is replaced with 5, because 2 was the first in.
Fault #2 (F): 3 is replaced with 2, because 3 was current first in.
Fault #3 (F): 1 is repalced with 4, because 1 was current first in.
continue until completion...

*Clock Policy
-Requires the association of an additional bit with each frame
*referred to as the use bit
-When a page is first loaded in memory or referenced, the use bit is set to 1
-the set of frames is considered to be a circular buffer
-Any frame with a use bit of 1 is passed over by the algorithm
-page frames visualized as laid out in a circle

~how it works~
the '*' denotes that the use speed = 1.
the '->' is the pointer pointing to the next frame (moves in circular list).
for the first fault, we need to find a frame with use speed = 0.
Example: all frames use speed = 1 on first fault, once the pointer moves the frame will be equal to 0 again. That is why on the first fault, 5 replaces 2.
NOTE: on fault #4, 2 is already being used...so when running the replacement policy for adding 2, the use speed is set to 1 denoted '*', but the pointer does not move because the pointer only moves when a new element is added.

Clock Policy Example

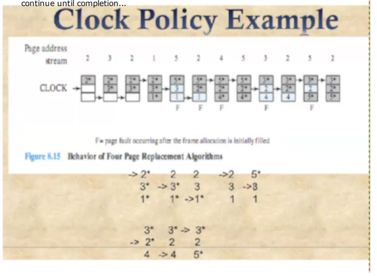


Figure 8.15 Behavior of Four Page Replacement Algorithms

*Page Buffering
-Improves paging performance and allows the use of a simpler page replacement policy

A replaced page is not lost, but rather assigned to one of two lists:
|
1)Free page list:
|
list of page frames available for reading in pages.

2)Modified page list:
|
pages are written out in clusters

*Replacement policy and cache size
-With large caches, replacement of pages can have a performance impact
*if the page frame selected for replacement is in the cache, that cache block is lost as well as the page that it holds
*in systems using page buffering, cache performance can be improved with a policy for page placement in the page buffer
*most operating systems place pages by selecting an arbitrary page from the page buffer
*Resident Set Management - part of the process that is in main memory
-The OS must decide how many pages to bring into main memory
-the smaller the amount of memory allocated to each process, the more processes can reside in memory
-small number of pages loaded increases page faults
-beyond a certain size, further allocations of pages will not effect the page fault rate.

-**Fixed-allocation**
*gives a process a fixed number of frames in main memory within which to execute
-when a page fault occurs, one of the pages of that process must be replaced

*Local Replacement
-number of frames allocated to a process is fixed.
-Page to be replaced is chosen from among the frames allocated to that process
*Global Replacement
-Not possible

-**Variable-allocation**
*allows the number of page frames allocated to a process to be varied over the lifetime of the process

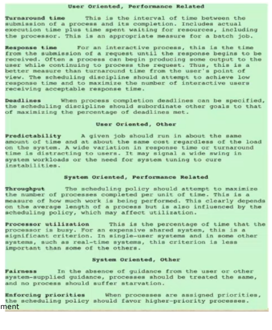
*Local Replacement
-The number of frames allocated to a process may be changed from time to time to maintain the working set of the process
-Page to be replaced is chosen from among the frames allocated to that process
*Global Replacement
-Page to be replaced is chosen from all available frames in main memory.

*Replacement Scope
-The scope of a replacement strategy can be categorized as global or local
*both types are activated by a page fault when there are no free page frames
Local: chooses only among the resident pages of the process that generated the page fault
Global: considers all unlocked pages in main memory
*Fixed Allocation, Local Scope
-Necessary to decide ahead of time the amount of allocation to give a process
-If allocation is too small, there will be a high page fault rate
-If allocation is too larger, there will be too few programs in main memory
*increased processor idle time
*increased time spent in swapping
*Variable Allocation, Global Scope
-Easiest to implement
*adopted in a number of operating systems
-OS maintains a list of free frames
-free frame is added to resident set of process when a page fault occurs
-If no frames are available the OS must choose a page currently in memory
-One way to counter potential problems is to use page buffering
*Variable Allocation, Local Scope
-When a new process is loaded into main memory, allocate to it a certain number of page frames as its resident set
-When a page fault occurs, select the page to replace from among the resident set of the process that suffers the fault
-Reevaluate the allocation provided to the process and increase or decrease it to improve overall performance
-Decision to increase or decrease a resident set size is based on the assessment of the likly future demands of active processes.
Key elements: -criteria used to determine resident set size. -the timing of changes
*Page Fault Frequency (PFF)
-Requires a use bit to be associated with each page in memory
-Bit is set to 1 when that page is accessed
-When a page fault occurs, the OS notes the virtual time since the last page fault for that process
-Does not perform well during the transient periods when there is a shift to a new locality
*Variable-interval Sampled Working Set (VSWS)
-Evalulates the working set of a process at sampling instances based on elapsed virtual time
-Driven by three parameters:
1)The minimum duration of the sampling interval
2)The maximum duration of the sampling interval
3)The number of page faults that are allowed to occur between sampling instances
*Cleaning Policy
-Concerned with determining when a modified page should be written out to seconday memory
-Demand Cleaning: a page is written out to seconday memory only when it has been selected for repalcement
-Precleaning: allows the writing of pages in batches
*Load Control
-Determines the number of processes that will be resident in main memory
*multiprogramming level
-Critical in effective memory management
-Too few processes, many occasions when all processes will be blocked and much time will be spent in swapping
-Too many processes will lead to thrashing

The pile
-Least complicated form of file organization
-Data are collected in the order they arrive
-Each record consists of one burst of data
-Purpose is simply to accumulate the mass of data and save it
-Record access is by exhaustive search
-Variable length, set of fields, chronological

Access Matrix-
The basic elements are:
subject – an entity capable of accessing objects
object – anything to which access is controlled
access right – the way in which an object is accessed by a subject
Access Control Lists - lists users and their permitted access rights
A matrix may be decomposed by columns, yielding access control lists
Capability Lists
Decomposition by rows yields capability tickets
A capability ticket specifies authorized objects and operations for a user

Turnaround time, Response time, Deadlines, Predictability, Throughput
Processor utilization, Fairness, Enforcing priorities,
(key words in image below for cntrl+f)


User Oriented, Performance Related

Turnaround time This is the interval of time between the submission of a process and its completion. Includes actual execution time plus time spent waiting for resources, including the processor. This is an appropriate measure for a batch job.

Response time For an interactive process, this is the time from the submission of a request until the response begins to be received. Often a process can begin producing some output to the user while continuing to process the request. Thus, this is a better measure than turnaround time from the user's point of view. The scheduling discipline should attempt to achieve low response time and to maximize the number of interactive users receiving acceptable response time.

Deadlines When process completion deadlines can be specified, the scheduling discipline should subordinate other goals to that of maximizing the percentage of deadlines met.

User Oriented, Other

Predictability A given job should run in about the same amount of time and at about the same cost regardless of the load on the system. A wide variation in response time or turnaround time is distracting to users. It may signal a wide swing in system workloads or the need for system tuning to cure instabilities.

System Oriented, Performance Related

Throughput The scheduling policy should attempt to maximize the number of processes completed per unit of time. This is a measure of how much work is being performed. This clearly depends on the average length of a process but is also influenced by the scheduling policy, which may affect utilization.

Processor utilization This is the percentage of time that the processor is busy. For an expensive shared system, this is a significant criterion. In single-user systems and in some other systems, such as real-time systems, this criterion is less important than some of the others.

System Oriented, Other

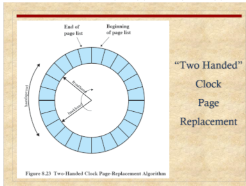
Fairness In the absence of guidance from the user or other system-supplied guidance, processes should be treated the same, and no process should suffer starvation.

Enforcing priorities When processes are assigned priorities, the scheduling policy should favor higher-priority processes.

Indexed Sequential File
-Adds an index to the file to support random access
-Adds an overflow file
-Greatly reduces the time required to access a single record
-Multiple levels of indexing can be used to provide greater efficiency in access

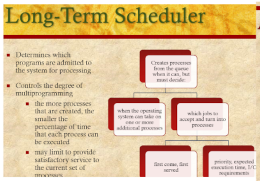
*Process Suspension
-If the degree of multiprogramming is to be reduced, one or more of the currently resident processes must be swapped out.
Six exsist: 1) Lowest-priority process, 2) Faulting process, 3) Last process activated, 4) Process with the smallest resident set, 5) Largest process, 6) Process with the largest remaining exe. window.

*Replacement policy for Unix system (srv4)
"Two Handed Clock Page Replacement"
Front hand: when the front hand moves through frames, the use speed is reset to 0.
Back hand: The first frame the backhand comes across that is 0 will be the frame that is replaced.



“Two Handed” Clock Page Replacement

Figure 8.23 Two-Handed Clock Page-Replacement Algorithm


Long-Term Scheduler

- Determines which programs are admitted to the system for processing
- Controls the degree of multiprogramming
  - the more processes that are created, the smaller the percentage of time that each process can be executed
  - may limit to provide satisfactory service to the current set of processes

Creates processes from the queue when it can, but must decide:
when the operating system can take on one or more additional processes
which jobs to accept and turn into processes
first come, first served
priority, expected execution time, I/O requirements

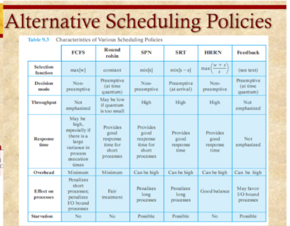

Alternative Scheduling Policies

Table 9.3 Characteristics of Various Scheduling Policies

	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

*Long-term scheduling - The decision to add to the pool of processes to be executed
-Determines which programs are admitted to the system for processing
-Controls the degree of multiprogramming
*The more processes that are created, the smaller the percentage of time that each process can be executed
*May lmit to provide satisfactory service to the current set of processes
*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
-Swapping-in decisions are based on the need to manage the degree of multiprogramming
*considers the memory requirements of the swapped-out processes
*Short-term scheduling - The decision as to which available process will be executed by the processor
-Known as the dispatcher
-Executes most frequently
-Makes the fine-grained decision of which process to execute next
-Invoked when an event occurs that may lead to the blocking of the current process or that may provide an opportunity to preempt a currently running process in favor of another
*Examples: Clock interrupts, I/O interrupts, Operating system calls, Signals(semaphores)
-Main objective is to allocate processor time to optimize certain aspects of system behavior
-A set of criteria is needed to evaluate the scheduling policy

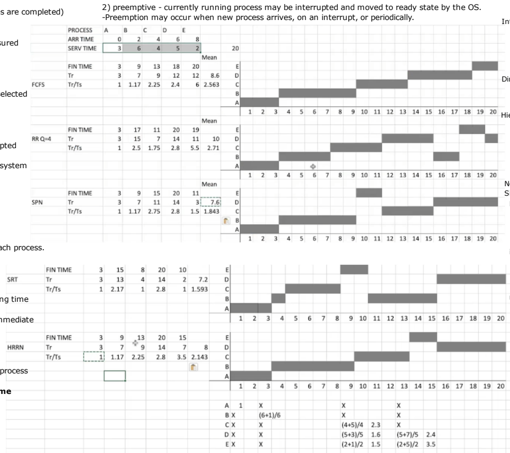
~User-oriented criteria
*relate to the behavior of the system as perceived by the individual user or process (such as response time in an interactive system). -important on virtually all systems.
~System-oriented criteria
*focus in on effective and efficient utilization of the processor(rate at which processes are completed)
*generally of minor importance on single-user systems
-Criteria can be classified into:
1) Performance-related (e.g. response time, throughput): quantitative & easily measured
2) Non-performance related(e.g. predictability): qualitative & hard to measure
*First-come-first-served (FCFS)
-Simplest scheduling policy
-Also known as first-in-first-out (FIFO) or a strict queing scheme
-When the current process ceases to execute, the longest process in the Ready queue is selected
-performs much better for long processes than short ones
-Tends to favor processor-bound processes over I/O-bound processes
*Round Robin
-Uses preemption based on a clock
-Also known as time slicing because each process is given a slice of time before being preepted
-Principal design issue is the length of the time quantum, or slice, to be used
-Particularly effective in a general-purpose time-sharing system or transaction processing system
-One drawback is its relative treatment of processor-bound and I/O-bound processes
Round Robin Execution - q = 1
*Shortest Process Next (SPN)
-Nonpreemptive policy in which the process with the shortest expected processing time is selected next
-A short process will jump to the head of the queue
-Possibility of starvation for longer processes
-One difficulty is the need to know, or at least estimate, the required processing time of each process.
-If the programmer's estimate is substantially under the actual running time, the system may abort the job.
*Shortest Remaining Time (SRT)
-Preemptive version of SPN
-Scheduler always chooses the process that has the shortest expected remaining processing time
-Risk of starvation of longer processes
-Should give superior turnaround time performance to SPN because a short job is given immediate preference to a running longer job
*Highest Response Ratio Next (HRRN)
-Chooses next process with the greatest ratio
-Attractive beacuse it accounts for the age of the process
-While shorter jobs are favored, aging without service increases the ratio so that a longer process will eventually get past competing shorter jobs

Ratio = (time spent waiting + expected service time) / expected service time

***Fair-Share Scheduling**
-Scheduling decisions based on the process sets
-Each user is assigned a share of the processor
-Objective is to monitor usage to give fewer resources to users who have had more than their fair share and more to those who have had less than their fair share.

*I/O scheduling - The decision as to which process's pending I/O request shall be handled by an available I/O device.
*Selection Function
-Determines which process, among ready processes, is selected next for execution
-May be based on priority, resource requirements, or the execution characteristics of the process
-If based on execution characteristics then important quantities are:
*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; generally, this quantity must be estimated or supplied by the user
*Decision Mode
*Specifies the instants in time at which the selection function is exercised
*Two categories:
1) nonpreemptive - once a process is in the running state, it will continue until it terminates or blocks itself for I/O

2) preemptive - currently running process may be interrupted and moved to ready state by the OS.
-Preemption may occur when new process arrives, on an interrupt, or periodically.



	PROCESS	A	B	C	D	E	Mean
	ARR TIME	0	2	4	6	8	
	SERV TIME	3	6	4	5	2	20
	FIN TIME	3	9	13	18	20	
	Tr	3	7	9	12	12	8.6
FCFS	Tr/Ts	1	1.17	2.25	2.4	6	2.563
	FIN TIME	3	17	11	20	19	
RR Q=4	Tr	3	15	7	14	11	10
	Tr/Ts	1	2.5	1.75	2.8	5.5	2.71
	FIN TIME	3	9	15	20	11	
SPN	Tr	3	7	11	14	3	7.6
	Tr/Ts	1	1.17	2.75	2.8	1.5	1.843
	FIN TIME	3	15	8	20	10	
SRT	Tr	3	13	4	14	2	7.2
	Tr/Ts	1	2.17	1	2.8	1	1.593
	FIN TIME	3	9	13	20	15	
HRRN	Tr	3	7	9	14	7	8
	Tr/Ts	1	1.17	2.25	2.8	3.5	2.143

A	1	X	X		X	
B	X	(6+1)/6	X		X	
C	X	X	(4+5)/4	2.3	X	
D	X	X	(5+3)/5	1.6	(5+7)/5	2.4
E	X	X	(2+1)/2	1.5	(2+5)/2	3.5

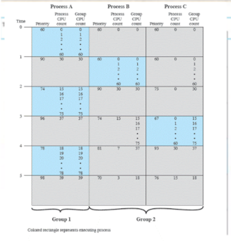


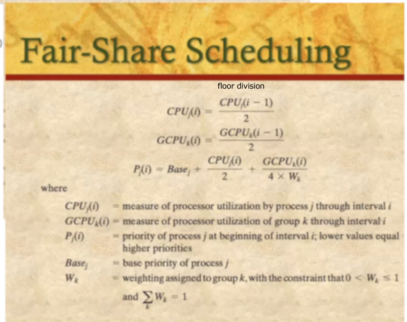
Figure 9.16 Example of Fair Share Scheduler—Three Processes, Two Groups

In the event the values are the same, lowest PID will be selected.
To find priority for time 1 (90), you take the original priority at time 0 (60) + the floor('Process CPU count' / 2) + floor(Group CPU count)

60 + floor(30/2) + floor(30/2) = 90

To find the Process CPU count and the Group CPU count you take the prvious process's value 'x' and use floor division by 2.
floor(x/2)
**in the example we get 30 by taking floor(60/2)

*The process with the lowest value will have the highest priority and will be executed first.


Fair-Share Scheduling

floor division

$$CPU_j(i) = \frac{CPU_j(i-1)}{2}$$

$$GCPU_k(i) = \frac{GCPU_k(i-1)}{2}$$

$$P_j(i) = Base_j + \frac{CPU_j(i)}{2} + \frac{GCPU_k(i)}{4 \times W_k}$$

where

$CPU_j(i)$ = measure of processor utilization by process j through interval i
$GCPU_k(i)$ = measure of processor utilization of group k through interval i
$P_j(i)$ = priority of process j at beginning of interval i; lower values equal higher priorities
$Base_j$ = base priority of process j
W_k = weighting assigned to group k, with the constraint that $0 < W_k \leq 1$ and $\sum_k W_k = 1$

*Bands - Used to optimize access to block devices and to allow the operating system to respond quickly to system calls.
*in decreasing order of priority the bands are: 1)swapper, 2)Block I/O device control, 3) File manipulation, 4) Character I/O device control, 5) User processes
*Classifications of Multiprocessor Systems
-Loosely coupled or distributed multiprocessor, or cluster
*consists of a collection of relatively autonomous systems, each processor having its own main memory and I/O channels
-Functionally specialized processors
*there is a master, general-purpose processor; specialized processors are controlled by the master processor and provide services to it.
-Tightly coupled multiprocessor
*consists of a set of processors that share a common main memory and are under the integrated control of an operating system
*Synchronization Granularity and Processes
Grain Size: Fine, Description - Parallelism inherent in a single instruction stream, Synchronization Interval (instruction): < 20
Grain Size: Medium, Description - Parallel processing or multitasking within a single application, Synchronization Interval (instruction): 20-200
Grain Size: Coarse, Description - Mutliprocessing of concurrent processes in a multiprogramming environment, Synchronization Interval (instruction): 200-2000
Grain Size: Very Coarse, Description - Distributed processing across network nodes to form a single computing environment , Synchronization Interval (instruction): 2000-1m
Grain Size: Independent, Description - Multiple unrelated processes, Synchronization Interval (instruction): not applicable
*Independent Parallelism
-no explicit synchronization among processes.
*each represent a separate, independent application or job
-Typical use is in a time-sharing system
-each use is performing a particular application, multiprocessor provides the same service as a multiprogrammed uniprocessor, because more than once processor is available, average response time to the users will be less.
*Coarse and Very Coarse-grained parallelism
-Synchronization among processes, but at a very gross level.
-Good for concurrent processes running on a multiprogrammed uniprocessor
*can be supported on a multiprocessor with little or no change to user software.
*Medium-Grained Parallelism
-Single application can be effectively implemented as a collection of threads within a single process.
*programmer must explicitly specify the potential parallelism of an application
*there needs to be a high degree of coordination and interaction among the threads of an application, leading to a medium-grain level of synchronization
-Because the various threads of an application interact so frequently, scheduling decisions concerning one thread may affect the performance of the entire application
*Fine-Grained Parallelism
-Represents a much more complex use of parallelism than is found in the use of threads
-Is a specialized and fragmented area with many different approaches
*Design Issues
-Thre approach taken will depend on the degree of granularity of applications and thenumber of processors available.
-Scheduling on a multiprocessor involves three interrelated issues: 1)actual dispatching of a process, 2)use of multiprogramming on individual processors, 3)assignment of processes to CPU's
*Assignment of Processes to Processors
-Assuming all processors are equal, it is simplest to treat processors as a pooled resource and assign processes to processors on demand ->static or dynamic needs to be determined.
-If a processor is permanently assigned to one processor from activation until its completion, then a dedicated short-term queue is maintained for each processor -> advantage is that there may be less overhead in the scheduling function, -> allows group or gang scheduling
-A disadvantage of static assignment is that one processor can be idle, with an empty queue, while another processor has a backlog
*to prevent this situation, a common queue can be used. *another option is dynamic load balancing.
-Approaches:
1)master/slave
*Key kernel functions always run on a particular processor, *Master is responsible for scheduling, *Slave sends service request to the master, *Is simple and requires little enhancement to a uniprocessor multiprogramming operating system, *Confilct resolution is simplified because one processor has control of all memory and I/O Resources.
-Disadvantages: failure of master brings down whole system, Master can become a performance bottleneck

2)Peer
*Kernel can execute on any processor, *Each processor does self-scheduling from the pool of available processes.
-Disadvantages: Operating system must ensure that two processors do not choose the same process and that the proceses are not somehow lost from queue.

*Process Scheduling
-Usually processes are not dedicated to processors
-A single queue is used for all processors
-if some sort of priority scheme is used, there are multiple queues based on priority
-System is viewed as being a multi-server queuing architecture
*Thread Scheduling
-Thread execution is separated from the rest of the definition of a process
-An application can be a set of threads that cooperate and execute concurrently in the same address space
-On a uniprocessor, threads can be used as a program structuring aid and to overlap I/O with processing
-In a multiprocessor system threads can be used to exploit true parallelism in an application
-Dramatic gains in performance are possible in multi-processor systems.
-small differences in thread management and scheduling can have an impact on applications that require significant interaction among threads.
*Approaches to Thread Scheduling
1)Load Sharing - processes are not assigned to a particular processor
-Simplest approach and carries over most directly from a uniprocessor environment
-**Advantages**: -load is dristributed evenly across the processor, -no centralized scheduler required, -the global queue can be organized and accessed using any of the schmeses discussed
-Versions of Load Sharing: 1)first-come-first-served, 2) smallest number of threads first, 3)preemptive smallest number of threads first
-**Disadvantages**: -Central queue occupies a region of memory that must be accessed in a manner that enforces mutual exclusion (bottlenecks), -Preemptive threads are unlikely to resume execution on the same processor (caching less efficient), -If all threads are treated as a common pool of threads, it is unlikely that all of the threads of a program will gain access to processors at the same time(process switches involved may seriously compromise performance.)
2)Gang Scheduling - a set of related thread scheduled to run on a set of processors at the same time, on a one-to-one basis
-Simultaneous scheduling of the threads that make up a single process
-Benefits: *synchronization blocking may be reduced, less process switching may be necessary, and performance will increase. *scheduling overhead may be reduced.
-Useful for medium-grained to fine-grained parallel applications whose performance severely degrades when any part of the application is not running while other parts are ready to run.
-Also beneficial for any parallel application
3)Dedicated Processor Assignment - provides implicit scheduling defined by the assignment of threads to processors
-When an application is scheduled, each of its threads is assigned to a processor that remains dedicated to that thread until the application runs to completion
-If a thread of application is blocked waiting for I/O or for synchronization with another thread, then that thread's processor remains idle(there is no multiprogramming of processors).
-Defense: *in a highly parallel system, with tens or hundreds of processors, processor utilization is no longer so important as a metric for effectiveness or performance. *the total avoidance of process switching during the lifetime of a program should result in a substantial speedup of that program.
4)Dynamic scheduling - the number of threads in a process can be altered during the course of execution
-For some applications it is possible to provide language and system tools that permit the number of threads in the process to be altered dynamically (will allow the OS to adjust load)
-Both the OS and the application are involved in making scheduling decisions.
-The scheduling responsibility of the OS is primarily limited to processor allocation.
-This approach is superior to gang scheduling or dedicated processor assignment for applications that can take advantage of it.
*Real-Time Systems
-The OS, and in particular the scheduler, is perhaps the most important component.
-Correctness of the system depends not only on the logical result of the computation but also on the time at which the results are produced
-Tasks or processes attempt to control or react to events that take place in the outside world.
-These events occur in "real time" and tasks must be able to keep up with them.
*Hard and Soft Real-Time Tasks
-Hard real-time task: 1) one that must meet its deadline. 2)otherwise it will cause unacceptable damage or a fatal error to the system
-Soft real-time task: 1) has an associated deadline that is desirable but not mandator. 2) it still makes sense to schedule and complete the task even if it has passed its deadline
**Periodic and Aperiodic Tasks
-Periodic tasks: requirement may be stated as: *once per period T, *exactly T units apart.
-Aperiodic tasks: *has a deadline by which it must finish or start. *may have a constraint on both start and finish time

Characteristics of Real Time Systems:
*Determinism
-Concerned with how long an operating system delays before acknowledging an interrupt
-Operations are performed at fixed, predetermined times or within predetermined time intervals (when multiple processes are competing for resources and processor time, no system will be fully deterministic).
Deterministic requests depend on: 1)the spped with which it can respond to interrupts. 2) Whether the system has suff. capacity to handle all requests within time.
*Responsiveness
-Together with determinism make up the response time to external events (critical for real-time systems that must meet timing requirements imposed by individuals, devices, data flows)
-Concerned with how long, after acknowledgment, it takes an OS to service the interrupt.
-Responsiveness includes: 1)amount of time required to initially handle the interrupt and begin exe of ISR. 2) amount of time required to perform the ISR. 3) effect of interrupt nesting
*User Control
-Generally much broader in a real-time OS than in ordinary OS.
-it is essential to allow the user fine-grained control over task priority
-User should be able to distinguish between hard and soft tasks and to specify relative priorities within each class.
-user may specify characteristics: 1)paging or process swapping, 2)what processes must always be resident in main mem. 3)disk transfer algo. 4)rights the processes in various priority bands
*Reliability
-More important for real-time systems than non-real time systems.
-Real-time systems respond to and control events in real time so loss or degredation of performance may have catastrophic consequences such as:financial/life loss, equipment damage
*Fail-Soft operation
-A characteristic that refers to the ability of a system to fail in such a way as to preserve as much capability and data as possible.
-Important aspect is stability. -a real-time system is stable if the system will meet the deadlines of its most critical, highest-priority tasks even if some less critical task deadlines are not met.

*Real-Time Scheduling
-Scheduling approaches depend on:
1)wheather a system performs schedulability analysis
-if it does, whether it is done statically or dynamically
2)Whether the result of the analysis itself produces a scheduler plan according to which tasks are dispatches at run time.
*Classes of real-time scheduling Algorithms
-Static table-driven approaches: 1)performs static analysis of feasible schedules of dispatching. 2) result is a schedule that determines, at run time, when a task must begin execution.
-Static priority-driven preemptive approaches: 1) a static analysis is performed but no schedule is drawn up. 2) analysis isused to assign priorities to tasks so that a traditional priority-driven preeptime scheduler can be used.
-Dynamic planning-based approaches:1)feasibility is determined at run time rather than offline prior to the start of exectuins. 2)result of the analysis is a schedule or plan used to dispatch task
-Dynamic best effort approaches: 1)no feasibility analysis is performed. 2) system tried to meet all deadlines and aborts any started process whose deadline is missed.
*Deadline Scheduling
-Real-time OS are designed with the objective of starting real-time tasks as rapdily as possible and emphasize rapid interrupt handling and task dispatching
-Real-time applications are generally not concerned with sheer speed but rather with completing(or starting) tasks at the most valuable times.
-Priorities provide a crude tool and do not capture the requirement of completion(or initiation) at the most valuable time
Information Used for Deadline Scheduling
-Ready time - time task becomes ready for execution
-Resource Requirements - resources required by the task while it is executing
-Starting Deadline - time task must begin
-Completion Deadline - time task must be completed
-Processing time - time required to execute the task to completion
-Priority - measures relative importance of the task
-Subtask scheduler - a task may be decomposed into a mandatory subtask and an optional subtask

Evolution of I/O Function
1)Processor directly controls a peripheral device, 2)A controller or I/O module is added, 3)Same configuration as step 2, but now interrupts are employed, 4)The I/O module is given direct control of memory via DMA, 5)The I/O module is enhanced to become a separate processor, with a specialized instruction set tailored for I/O, 6)The I/O module has a local memory of its own and is, in fact, a computer in its own right

*Categories of I/O Devices
External devices that engage in I/O with computer systems can be grouped into three categories:
**Human readable
-suitable for communicating with the computer user
-printers, terminals, video display, keyboard, mouse
**Machine readable
-suitable for communicating with electronic equipment
-disk drives, USB keys, sensors, controllers
**Communication
-suitable for communicating with remote devices
-modems, digital line drivers
Programmed I/O
-the processor issues an I/O command on behalf of a process to an I/O module; that process then busy waits for the operation to be completed before proceeding
- No interrupts, I/O to memory transfer through processor (alongside interrupt)
Interrupt-driven I/O
-the processor issues an I/O command on behalf of a process
if non-blocking – processor continues to execute instructions from the process that issued the I/O command
if blocking – the next instruction the processor executes is from the OS, which will put the current process in a blocked state and schedule another process
Direct Memory Access (DMA)
-a DMA module controls the exchange of data between main memory and an I/O module; Direct I/O to memory transfer
Hierachical Design - multiple layers that have their own tasks (layers close to user, to hardware)
-Functions of the operating system shuld be separated according to their complexity, their characteristic time scale, and their level of abstraction
-Leads to an organization of the operating system into a series of layers
-Each layer performs a related subset of the functions required of the operating system
-Layers should be defined so that changes in one layer do not require changes in other layers
No buffer- Without a buffer, the OS directly accesses the device when it needs
Single buffer- Operating system assigns a buffer in main memory for an I/O request
Block-Oriented Single Buffer
-Input transfers are made to the system buffer
-Reading ahead/anticipated input
is done in the expectation that the block will eventually be needed
when the transfer is complete, the process moves the block into user space and immediately requests another block
-Generally provides a speedup compared to the lack of system buffering
-Disadvantages:
complicates the logic in the operating system
swapping logic is also affected
Utility of Buffering
-Technique that smoothes out peaks in I/O demand
with enough demand eventually all buffers become full and their advantage is lost
-When there is a variety of I/O and process activities to service, buffering can increase the efficiency of the OS and the performance of individual processes
Disk scheduling
*First-In, First-Out (FIFO)
-Processes in sequential order
-Fair to all processes
-Approximates random scheduling in performance if there are many processes competing for the disk
*Shortest Service Time First (SSTF)
-Select the disk I/O request that requires the least movement of the disk arm from its current position
-Always choose the minimum seek time
*SCAN
-Also known as the elevator algorithm
-Arm moves in one direction only
satisfies all outstanding requests until it reaches the last track in that direction then the direction is reversed
-Favors jobs whose requests are for tracks nearest to both innermost and outermost tracks
*C-SCAN (Circular Scan)
-Restricts scanning to one direction only
-When the last track has been visited in one direction, the arm is returned to the opposite end of the disk and the scan begins again
*N-Step-SCAN
-Segments the disk request queue into subqueues of length N
-Subqueues are processed one at a time, using SCAN
-While a queue is being processed new requests must be added to some other queue
-If fewer than N requests are available at the end of a scan, all of them are processed with the next scan
*FSCAN
-Uses two subqueues
-When a scan begins, all of the requests are in one of the queues, with the other empty
-During scan, all new requests are put into the other queue
-Service of new requests is deferred until all of the old requests have been processed
Disk cache
-Cache memory is used to apply to a memory that is smaller and faster than main memory and that is interposed between main memory and the processor
-Reduces average memory access time by exploiting the principle of locality
-Disk cache is a buffer in main memory for disk sectors
-Contains a copy of some of the sectors on the disk
Files
-Data collections created by users
-The File System is one of the most important parts of the OS to a user
-Desirable properties of files:
Long-term existence- files are stored on disk or other secondary storage and do not disappear when a user logs off
Sharable between processes-files have names and can have associated access permissions that permit controlled sharing
Structure- files can be organized into hierarchical or more complex structure to reflect the relationships among files
File Management System Objectives
-Meet the data management needs of the user
-Guarantee that the data in the file are valid
-Optimize performance
-Provide I/O support for a variety of storage device types
-Minimize the potential for lost or destroyed data
-Provide a standardized set of I/O interface routines to user processes
-Provide I/O support for multiple users in the case of multiple-user systems
Basic I/O Supervisor
-Responsible for all file I/O initiation and termination
-Control structures that deal with device I/O, scheduling, and file status are maintained
-Selects the device on which I/O is to be performed
-Concerned with scheduling disk and tape accesses to optimize performance
-I/O buffers are assigned and secondary memory is allocated at this level
-Part of the operating system
Logical I/O
Enables users and applications to access records ->Provides general-purpose record I/O capability ->Maintains basic data about file
Access Method
-Level of the file system closest to the user
-Provides a standard interface between applications and the file systems and devices that hold the data
-Different access methods reflect different file structures and different ways of accessing and processing the data

File Organization Types
The pile, the sequential file, the indexed sequential file, the direct, or hashed file, the indexed file

File Organization and Access
-File organization is the logical structuring of the records as determined by the way in which they are accessed
-In choosing a file organization, several criteria are important:
short access time
ease of update
economy of storage
simple maintenance
reliability
-Priority of criteria depends on the application that will use the file
B-Trees
-A balanced tree structure with all branches of equal length
-Standard method of organizing indexes for databases
-Commonly used in OS file systems
-Provides for efficient searching, adding, and deleting of items
Characteristics:
-the tree consists of a number of nodes and leaves
-each node contains at least one key which uniquely identifies a file record, and more than one pointer to child nodes or leaves
-each node is limited to the same number of maximum keys
-the keys in a node are stored in non-decreasing order; each node has one more pointer than keys
-every node has at most 2d – 1 keys and 2d children or, equivalently, 2d pointers
-every node, except for the root, has at least d – 1 keys and d pointers, as a result, each internal node, except the root, is at least half full and has at least d children
-the root has at least 1 key and 2 children
-all leaves appear on the same level and contain no information. This is a logical construct to terminate the tree; the actual implementation may differ.
-a nonleaf node with k pointers contains k – 1 keys
File Allocation
-On secondary storage, a file consists of a collection of blocks
-The operating system or file management system is responsible for allocating blocks to files
-The approach taken for file allocation may influence the approach taken for free space management
-Space is allocated to a file as one or more portions (contiguous set of allocated blocks)
-File allocation table (FAT): data structure used to keep track of the portions assigned to a file
Preallocation:
A preallocation policy requires that the maximum size of a file be declared at the time of the file creation request
For many applications it is difficult to estimate reliably the maximum potential size of the file
tends to be wasteful
Dynamic allocation- allocates space to a file in portions as needed

Earliest Deadline scheduling (Earliest Deadline First Algorithm)
--->>
-Select earliest deadline and run it, check as processes arrive
-ex. A1 and B1 arrive at 0, A1 deadline = 20, B1 = 50, A1 executes first
-Uniprocessor System
*Rate Monotonic Scheduling - higher rate/-low period, higher priority; lower period/cycle = higher rate
*RMS Upper bound- underneath this, can perform RMS

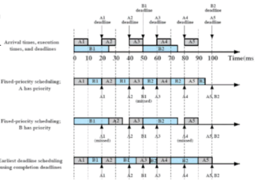

A1 deadline
A2 deadline
B1 deadline
A3 deadline
A4 deadline
B2 deadline
A5 deadline
Arrival times, execution times, and deadlines
0 10 20 30 40 50 60 70 80 90 100 Time(ms)
Fixed-priority scheduling; A has priority
Fixed-priority scheduling; B has priority
Earliest deadline scheduling using completion deadlines
A1 B1 A2 A3 A4 B2 A5
(missed)

Figure 10.6 Scheduling of Periodic Real-time Tasks with Completion Deadlines (based on Table 10.2)

Priority Inversion
-Can occur in any priority-based preemptive scheduling scheme
-Particularly relevant in the context of real-time scheduling
-Best-known instance involved the Mars Pathfinder mission
-Occurs when circumstances within the system force a higher priority task to wait for a lower priority task
*Unbounded Priority Inversion
-the duration of a priority inversion depends not only on the time required to handle a shared resource, but also on the unpredictable actions of other unrelated tasks
*Priority Inheritance - lower-priority task inherits priority of higher-priority task that waits on shared resource
Differences in I/O Devices
Data Rate- there may be differences of magnitude between the data transfer rates
Application- the use to which a device is put has an influence on the software
Complexity of Control- the effect on the operating system is filtered by the complexity of the I/O module that controls the device
Unit of Transfer-data may be transferred as a stream of bytes or characters or in larger blocks
Data Representation-different data encoding schemes are used by different devices
Error conditions-the nature of errors, the way in which they are reported, their consequences, and the available range of responses differs from one device to another
Design Objectives
*Efficiency
-Major effort in I/O design
-Important because I/O operations often form a bottleneck
-Most I/O devices are extremely slow compared with main memory and the processor
-The area that has received the most attention is disk I/O
*Generality
-Desirable to handle all devices in a uniform manner
-Applies to the way processes view I/ O devices and the way the operating system manages I/O devices and operations
-Diversity of devices makes it difficult to achieve true generality
-Use a hierarchical, modular approach to the design of the I/O function
***Buffering**- Perform input transfers in advance of requests being made and perform output transfers some time after the request is made
**Block-oriented device
-stores information in blocks that are usually of fixed size
transfers are made one block at a time
possible to reference data by its block number
disks and USB keys are examples
**Stream-oriented device
-transfers data in and out as a stream of bytes
-no block structure
-terminals, printers, communications ports, and most other devices that are not secondary storage are examples
*Stream-Oriented Single Buffer
**Line-at-a-time operation
-appropriate for scroll-mode terminals (dumb terminals)
-user input is one line at a time with a carriage return signaling the end of a line
-output to the terminal is similarly one line at a time
**Byte-at-a-time operation
-used on forms-mode terminals
-when each keystroke is significant
-other peripherals such as sensors and controllers
Double Buffer
-Use two system buffers instead of one
-A process can transfer data to or from one buffer while the operating system empties or fills the other buffer
-Also known as buffer swapping
Circular Buffer
-Two or more buffers are used
-Each individual buffer is one unit in a circular buffer
-Used when I/O operation must keep up with process
Positioning the Read/Write Heads
-When the disk drive is operating, the disk is rotating at constant speed
-To read or write the head must be positioned at the desired track and at the beginning of the desired sector on that track
-Track selection involves moving the head in a movable-head system or electronically selecting one head on a fixed-head system
-On a movable-head system the time it takes to position the head at the track is known as seek time
-The time it takes for the beginning of the sector to reach the head is known as rotational delay
-The sum of the seek time and the rotational delay equals the access time
RAID-Redundant Array of Independent Disks
-Consists of seven levels, zero through six
Design architectures share three characteristics
-RAID is a set of physical disk drives viewed by the operating system as a single logical drive
-redundant disk capacity is used to store parity information, which guarantees data recoverability in case of a disk failure
-data are distributed across the physical drives of an array in a scheme known as striping
RAID Level 0:
-Not a true RAID because it **does not include redundancy** to improve performance or provide data protection
-User and system data are distributed across all of the disks in the array
-Logical disk is divided into strips
RAID Level 1:
-Redundancy is achieved by the simple expedient of duplicating all the data
-There is no “write penalty”
-When a drive fails the data may still be accessed from the second drive
-Principal disadvantage is the cost
RAID Level 2:
-Makes use of a parallel access technique
-Data striping is used
-Typically a Hamming code is used
-Effective choice in an environment in which many disk errors occur
RAID Level 3:
-Requires only a single redundant disk, no matter how large the disk array
-Employs parallel access, with data distributed in small strips
-Can achieve very high data transfer rates
RAID Level 4 (Blocks):
-Makes use of an independent access technique
-A bit-by-bit parity strip is calculated across corresponding strips on each data disk, and the parity bits are stored in the corresponding strip on the parity disk
-Involves a write penalty when an I/O write request of small size is performed
RAID Level 5 (Blocks)-
-Similar to RAID-4 but distributes the parity bits across all disks
-Typical allocation is a round-robin scheme
-Has the characteristic that the loss of any one disk does not result in data loss
RAID Level 6 (Blocks)-
-Two different parity calculations are carried out and stored in separate blocks on different disks
-Provides extremely high data availability
-Incurs a substantial write penalty because each write affects two parity blocks
Least Frequently Used (LFU)
-The block that has experienced the fewest references is replaced
-A counter is associated with each block
-Counter is incremented each time block is accessed
-When replacement is required, the block with the smallest count is selected
File Systems
-Provide a means to store data organized as files as well as a collection of functions that can be performed on files
-Maintain a set of attributes associated with the file
-Typical operations include: Create, Delete, Open, Close, Read, Write
***File Structures Terms**
**Field
-basic element of data
-contains a single value
-fixed or variable length
**File
-collection of similar records
-treated as a single entity
-may be referenced by name
-access control restrictions usually apply at the file level
**Database
-collection of related data
-relationships among elements of data are explicit
-designed for use by a number of different applications
-consists of one or more types of files
**Record
-collection of related fields that can be treated as a unit
-fixed or variable length
Minimal User Requirements
1)should be able to create, delete, read, write and modify files
2)may have controlled access to other users’ files
3)may control what type of accesses are allowed to the files
4)should be able to restructure the files in a form appropriate to the problem
5)should be able to move data between files
6)should be able to back up and recover files in case of damage
7)should be able to access his or her files by name rather than by numeric identifier
Device Drivers
-Lowest level
-Communicates directly with peripheral devices
-Responsible for starting I/O operations on a device
-Processes the completion of an I/O request
-Considered to be part of the operating system
Basic File System
-Also referred to as the physical I/O level
-Primary interface with the environment outside the computer system
-Deals with blocks of data that are exchanged with disk or tape systems
-Concerned with the placement of blocks on the secondary storage device
-Concerned with buffering blocks in main memory
-Considered part of the operating system
The sequential file
-Most common form of file structure
-A fixed format is used for records, set of fields
-Key field uniquely identifies the record
-Typically used in batch applications
-Only organization that is easily stored on tape as well as disk
-Sequential order based on key field
Indexed File
-Records are accessed only through their indexes
-Variable-length records can be employed
-Exhaustive index contains one entry for every record in the main file
-Partial index contains entries to records where the field of interest exists
-Used mostly in applications where timeliness of information is critical
-Examples would be airline reservation systems and inventory control systems
Direct or Hashed File
-Access directly any block of a known address
-Makes use of hashing on the key value
-Often used where:
very rapid access is required
fixed-length records are used
records are always accessed one at a time
examples: directories, pricing tables, schedules, name lists
Operations performed on a directory: Search, Create files, Delete Files, List Directory, Update Directory
File Sharing
Two issues arise when allowing files to be shared among a number of users:
-access rights, management of simulatenous access
Two Level Scheme
-There is one directory for each user and a master directory
-Master directory has an entry for each user directory providing address and access control information
-Each user directory is a simple list of the files of that user
-Names must be unique only within the collection of files of a single user
-File system can easily enforce access restriction on directories
-Master directory with user directories underneath it
-Each user directory may have subdirectories and files as entries
Record Blocking-
-Blocks are the unit of I/O with secondary storage
-for I/O to be performed records must be organized as blocks
1) Fixed-Length Blocking – fixed-length records are used, and an integral number of records are stored in a block
Internal fragmentation – unused space at the end of each block
2) Variable-Length Spanned Blocking – variable-length records are used and are packed into blocks with no unused space
3) Variable-Length Unspanned Blocking – variable-length records are used, but spanning is not employed
Access Rights
None
the user would not be allowed to read the user directory that includes the file
Knowledge
the user can determine that the file exists and who its owner is and can then petition the owner for additional access rights
Execution
the user can load and execute a program but cannot copy it
Reading
the user can read the file for any purpose, including copying and execution
Appending
the user can add data to the file but cannot modify or delete any of the file’s contents
Updating
the user can modify, delete, and add to the file’s data
Changing protection
the user can change the access rights granted to other users
Deletion
the user can delete the file from the file system
Portion Size- In choosing a portion size there is a trade-off between efficiency from the point of view of a single file versus overall system efficiency
Items to be considered:
1) contiguity of space increases performance, especially for Retrieve_Next operations, and greatly for transactions running in a transaction-oriented operating system
2)having a large number of small portions increases the size of tables needed to manage the allocation information
3)having fixed-size portions simplifies the reallocation of space
4)having variable-size or small fixed-size portions minimizes waste of unused storage due to overallocation
Alternatives for portion sizes
Variable, large contiguous portions
provides better performance
the variable size avoids waste
the file allocation tables are small
A single contiguous set of blocks is allocated to a file at the time of file creation
Preallocation strategy using variable-size portions
Is the best from the point of view of the individual sequential file; compaction
Blocks
small fixed portions provide greater flexibility
they may require large tables or complex structures for their allocation
contiguity has been abandoned as a primary goal
blocks are allocated as needed
Chained Allocation
Allocation is on an individual block basis
Each block contains a pointer to the next block in the chain
The file allocation table needs just a single entry for each file
No external fragmentation to worry about
Best for sequential files
Indexed allocation with block portions or variable length portions
Free Space Management
Disk allocation table needed alongside file allocation table
Bit Tables: vector with one bit for each block on disk, 0 = free, 1 = used
Advantages: works with any file allocation method, small
Chained Free Portions:The free portions may be chained together by using a pointer and length value in each free portion
Negligible space overhead because there is no need for a disk allocation table
Suited to all file allocation methods
Disadvantages: fragmentation, every time you allocate a block you need to read the block first to recover the pointer to the new first free block before writing data to that block
Indexing- free space as file, use index table, variable-size, supports all allocation methods