CHAPTER 12 FILE MANAGEMENT

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 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 Structure
Four terms are commonly used when discussing files:
Field
Record
File
Database
Field
basic element of data
contains a single value
fixed or variable length
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
File
collection of similar records
treated as a single entity
may be referenced by name
access control restrictions usually apply at the file level
Record
collection of related fields that can be treated as a unit by some application program
fixed or variable length
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
Minimal User Requirements
Each user:
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
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 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
File Organization Types
Five of the common file organizations are:
The pile
The sequential file
The indexed sequential file
The indexed file
The direct, or hashed, file
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
The Sequential File
Most common form of file structure
A fixed format is used for records
Key field uniquely identifies the record
Typically used in batch applications
Only organization that is easily stored on tape as well as disk

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

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 are:
directories
pricing tables
schedules
name lists
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
B-Tree Characteristics
A tree structure (no closed loops) with the following 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 key
- the keys in a node are stored in non-decreasing order; each node has one more pointer than keys
B-Tree Characteristics
A B-tree is characterized by its minimum degree d and satisfies the following properties:
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
Operations Performed on a Directory
Search
Create files
Delete files
List directory
Update directory
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

Tree-Structured Directory
Master directory with user directories underneath it
Each user directory may have subdirectories and files as entries
File Sharing
Two issues arise when allowing files to be shared among a number of users:
access rights
management of simultaneous access
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

User Access Rights
Owner
usually the initial creator of the file
has full rights
may grant rights to others
Specific Users
individual users who are designated by user ID
User Groups
a set of users who are not individually defined
All
all users who have access to this system
these are public files
Record Blocking
Blocks are the unit of I/O with secondary storage
for I/O to be performed records must be organized as blocks
Given the size of a block, three methods of blocking can be used:
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
Variable-Length Spanned Blocking – variable-length records are used and are packed into blocks with no unused space
Variable-Length Unspanned Blocking – variable-length records are used, but spanning is not employed
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 vs Dynamic Allocation
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 because users and application programmers tend to overestimate size
Dynamic allocation allocates space to a file in portions as needed
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:
contiguity of space increases performance, especially for Retrieve_Next operations, and greatly for transactions running in a transaction-oriented operating system
having a large number of small portions increases the size of tables needed to manage the allocation information
having fixed-size portions simplifies the reallocation of space
having variable-size or small fixed-size portions minimizes waste of unused storage due to overallocation
Alternatives
Variable, large contiguous portions
provides better performance
the variable size avoids waste
the file allocation tables are small
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
Contiguous File Allocation
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
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
Free Space Management
Just as allocated space must be managed, so must the unallocated space
To perform file allocation, it is necessary to know which blocks are available
A disk allocation table is needed in addition to a file allocation table
Free Space Management
Just as allocated space must be managed, so must the unallocated space
To perform file allocation, it is necessary to know which blocks are available
A disk allocation table is needed in addition to a file allocation table
Bit Tables
This method uses a vector containing one bit for each block on the disk
Each entry of a 0 corresponds to a free block, and each 1 corresponds to a block in use
Advantages:
works well with any file allocation method
it is as small as possible
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:
leads to 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
Treats free space as a file and uses an index table as it would for file allocation
For efficiency, the index should be on the basis of variable-size portions rather than blocks
This approach provides efficient support for all of the file allocation methods

Free Block List
Each block is assigned a number sequentially
the list of the numbers of all free blocks is maintained in a reserved portion of the disk
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
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 FIFO queue, with a few thousand entries from both the head and the tail of the queue in main memory
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
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
A matrix may be decomposed by columns, yielding access control lists
The access control list lists users and their permitted access rights
Capability Lists
Decomposition by rows yields capability tickets
A capability ticket specifies authorized objects and operations for a user
UNIX File Management
Regular, or ordinary
contains arbitrary data in zero or more data blocks
Directory
contains a list of file names plus pointers to associated inodes
Special
contains no data but provides a mechanism to map physical devices to file names
Named pipes
an interprocess communications facility
Links
an alternative file name for an existing file
Symbolic links
a data file that contains the name of the file it is linked to
Inodes
All types of UNIX files are administered by the OS by means of inodes
An inode (index node) is a control structure that contains the key information needed by the operating system for a particular file
Several file names may be associated with a single inode
an active inode is associated with exactly one file
each file is controlled by exactly one inode
File Allocation
File allocation is done on a block basis
Allocation is dynamic, as needed, rather than using preallocation
An indexed method is used to keep track of each file, with part of the index stored in the inode for the file
In all UNIX implementations the inode includes a number of direct pointers and three indirect pointers (single, double, triple

Uniprocessor scheduling algorithms

Execute FCFS for the following group of processes and complete the following table:
Tfinish = cumulative sum of Ts
Tr = Tfinish - Tarrival

Process	A	B	C	D
Tarrival	0	2	4	6
Ts(service)	3	5	4	1
Tfinish	3	3+ 5 = 8	3 + 5 + 4 = 12	3 + 5 + 4 + 1 = 13
Tr	3 - 0 = 3	8 - 2 = 6	12 - 4 = 8	13 - 6 = 7

Execute RR (Q=4) for the following group of processes and complete the following table:

Process	A	B	C	D
Tarrival	0	2	4	6
Ts(service)	3	5	4	1
Tfinish	3	13	11	12
Tr	3 - 0 = 3	13 - 2 = 11	11 - 4 = 7	12 - 6 = 6

Queue: A, B, ~~C~~, ~~D~~, B

A: 3 - 3 = 0
B: 5 - 4 = 1 - 1 = 0
C: 4 - 4 = 0
D: 1 - 1 = 0

A: 3
B: 3 + 4 + 4 + 1 + 1 = 13
C: 3 + 4 + 4 = 11
D: 3 + 4 + 4 + 1 = 12

_Execute SPN for the following group of processes and complete the following table:

Process	A	B	C	D
Tarrival	0	1	5	6
Ts(service)	4	2	3	1
Tfinish	4	4 + 2 = 6	4 + 2 + 1 + 3 = 10	4 + 2 + 1 = 7
Tr	4 - 0 = 4	6 - 1 = 5	10 - 5 = 5	7 - 6 = 1

Notes:
For the process that have already arrived choose the one with the shortest service time

---

Execute SRT for the following group of processes and complete the following table:

Process	A	B	C	D	E
Tarrival	0	2	4	6	8
Ts(service)	2	3	~~5~~ → 4	1	4
Tfinish	2	2 + 3 = 5	~~5 + 1~~ → 7 + (1 + 3) = 11	5 + 1 + 1 = 7	11 + 4 = 15
Tr	2 - 0 = 2	5 - 2 = 3	11 - 4 = 7	7 - 6 = 1	15 - 8 = 7

Execute HRRN for the following group of processes and complete the following table:

Process	A	B	C	D	E
Tarrival	0	2	4	6	8
Ts(service)	2	3	5	1	4
Tfinish	2	5	10	11	15
Tr	2 - 0 = 2	5 - 2 = 3	10 - 4 = 6	11 - 6 = 5	15 - 8 = 7

Wait = (Current Time) - Arrival
Ratio = (Wait + ServiceTime) / ServiceTime

WaitD = 10 - 6 = 4; RatioD = (4 + 1) / 1 = 5
WaitE = 10 - 8 = 2 RatioE = (2 + 40 / 4 = 1.5

Notes:
-For as long as only one process is in the system at a time we don't have to follow any ratio rules
-Choose the process with the biggest ratio

File Systems
1 KiloByte = 1024 bytes
1 MegaByte = 1,048,576 bytes
1 GigaByte = 1,073,741,824 bytes

64 bit system and 4 KByte Block size example:
64 bit / 8 = 8 bytes
(4 KBytes Block size * 1024) = 4096(size of a block in bytes)

Level	Number of Blocks	Number of Bytes
Direct Level	12(given for every)	12 * 4096 = 49152
Single Indirect Level	4096 / 8 Bytes = 512	512 * 4096 = 2097152 Bytes or 2MBytes
Double Indirect Level	512^2 = 262144 or 256KBytes	(262144) * 4096 = 1073741824 or 1GBytes
Triple Indirect Level	512^3 = 134217728 or 128M	134217728 * 4096= 549755813888

Fair Share Algorithm

You can assume that:

	Group 1			Group 2		
Time	Process A			Process B		
	Priority	Process	Groupt	Priority	Process	Groupt
0	45	0	0	45	0	0
1	75	30	30	45	0	0
2	59	15	15	75	30	30

1. The base priority is equal to 45.
2. The processor is interrupted 60 times per time instant (the number of counts of the process that is currently running will be increased).
3. The weight of Group 1 is equal to the weight of Group 2.
4. If the priority of the two processes is the same, you will use the lowest PID criterion (using lexicographical order).

1 sec:
60/2 = 30
60/2 = 30
45 + (30/2) + (30/2) = 75

2 sec:
60/2 = 30
60/2 = 30
45 + (30/2) + (30/2) = 75

30/2 = 15
30/2 = 15
45 + floor(15/2) + floor(15/2) = 59

CHAPTER 7
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 be temporarily 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 be individually copied into main memory (combined segmentation and paging)
Memory management requirements - -Relocation
-Protection
-Sharing
-Logical organization
-Physical organization
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
Addressing Requirements -
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
Sharing - -Advantageous to allow each process access the same copy of the program rather than have their own separate copy
-Memory management must allow controlled access to shared areas of memory without compromising protection
-Mechanisms used to support relocation support sharing capabilities
logical organization - Memory is organized as linear
Segmentation is the tool that most readily satisfies requirements
Modules - Programs can be written in them
-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
Physical Organization - Cannot leave the programmer with the responsibility to manage memory.
Memory available for a program plus its data may be insufficient.
Programmer does not know how much space will be available.
Overlaying allows various modules to be assigned the same region of memory but is time consuming to program.
Memory Management - -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
Memory management techniques -
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
Weakness
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.
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
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.
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 necessary to load all of the pages of a process. Nonresident pages that are needed are brought in later automatically.
strengths
No external fragmentation, higher degree of multiprogramming large virtual address space
weaknesses:
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.
strengths:
No internal fragmentation, higher degree of multiprogramming; large virtual address space; protection and sharing support
weaknesses:
Overhead of complex memory management
Fixed-size partitioning 2 - -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 in the Ready or Running state
Disadvantages of fixed partitioning - -A program may be too big to fit in a partition
program needs to be designed with 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
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
Disadvantages of unequal size partitions - -The number of partitions specified as system generation time limits the number of active processes in the system
- Small jobs will not utilize partition space efficiently
Dynamic partitioning -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

-compaction and external fragmentation
External Fragmentation - -Memory becomes more more fragmented
- memory utilization declines.
Compaction - -technique for overcoming external fragmentation
-OS shifts processes to that they are contiguous
-free memory is together in one block
-time consuming and wastes CPU time
Placement Algorithms - Best-fit
First-fit
Next-fit
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
Buddy system - *comprised of fixed and dynamic partitioning schemes
*space available for allocation is treated as single block
memory blocks are available of size 2^k words, L<= K <= U
Addresses - Logical, Relative and Physical/Absolute

Logical Address - reference to a memory location independent of the current assignment of data to memory
relative address - address is expressed as a location relative to some known point
Physical or Absolute Address - 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
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
Segmentation - -A program can be subdivided into segments
-may very in length
-there is a maximum length
-Addressing consist of two parts:
-segment number
-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 of 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 shareable, then no other process should have access to the contents of that memory
Buffer overflow attacks - -Security threat related to memory management
-Also known as 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
Defending against buffer overflows - -Prevention
-Detecting and aborting
-Countermeasure categories:
Compile time defenses and Runtime defenses
Compile-time defenses - aim to harden programs to resist attacks in new programs
Run-Time Defenses - aim to detect and abort attacks in existing programs
Summary of Memory Management - -one of the most important and complex tasks of an operating system
-needs to be treated as a resource to be allocated and shared among a number of active processes
-desirable 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 pages
segmentation- pieces of varying size

CHAPTER 8 VIRTUAL MEMORY
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
-If these two characteristics are present, it is not necessary that all of the pages or segments of a process be in main memory during execution
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 location
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.
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
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 will 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
Characteristics of Paging and Segmentation -
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
-Avoid thrashing
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
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 includes - Page number
Process identifier- the process that owns 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 can cause two physical memory accesses:
-one to fetch the table entry
- one to fetch the data
To overcome the effect of doubling 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 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
page sizes example -
design issue of Page size - -the design issue of page size is related to the size of physical main memory and program size
-main memory is getting larger and address space used by applications is also growing
- most obvious on personal computers where applications are becoming increasingly complex

Contemporary programming - Contemporary programming techniques used in large programs tend to decrease the locality of references within a process
Segmentation - segmentation allows the programmer to view memory as consisting of multiple address space or segments
Advantages of segmentation - -simplifies handling of growing data structures
-allows program 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 main memory frame.
-Each segmentation is visible to the programmer
-Paging is transparent to the programmer
Protection 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
Operating system software - the design of memory management portion of an operating system depends on three fundamental areas of choice:
-whether or not to use virtual memory techniques
-the use of paging or segmentation or both
-the algorithms employed for various aspects of memory management
policies for virtual memory -
Fetch Policy - determines when a page should be brought into memory
two main types of fetch policy - Demand paging and prepaging
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.
Prepaging - -pages other than the one demanded by a page fault are brought in
-exploits the characteristics of most secondary memory of 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
Algorithms used for the selection of a page to replace - -Optimal
-Least recently used (LRU)
-First in first out (FIFO)
-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
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 at least likely to be referenced in the near future
-Difficult to implement
-one approach is to tag each page with the last time of reference
-this requires a great deal of overhead
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
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 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.
Page Buffering - Improves paging performance and allows the use of a simpler page replacement policy
Free Page list and Modified page list - A replaced page is not lost but rather assigned to one of these two lists
Free Page List - list of page frames available for reading in pages
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 it holds
-in the systems using page buffering, cache performance can be improved with a policy for page replacement in the page buffer
-most operating systems place pages by selecting an arbitrary page frame from the page buffer
- pages are written out in clusters
Resident Set Management - 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 increase page faults
-beyond a certain size, further allocations of pages will not effect the page fault rate
Resident Set Size - Fixed Allocation and Variable Allocation
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
Variable Allocation - Allows the number of page frames allocated to a process to be varied over the lifetime of the process.
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 Replacement scope - Chooses only among the resident pages of the process that generated the page fault
Global Replacement scope - considers all unlocked pages in main memory
Resident Set Management Summary -
Fixed Allocation, Local Scope - -Necessary to decide ahead of time the amount of allocation to give a process
-If the allocation is too small, there will be a high page fault rate
-If the allocation is too large, 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 the memory
-One way to counter potential problems is to use page buffering
Variable Allocation Local Scope - -Decision is to increase or decrease a resident size is based on the assessment of the likely future demands of active processes
key elements:
-criteria used to determine resident set size
-the timing of changes
Page Fault Frequency - -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 - -Evaluates the working set of a process at sampling instances based on elapsed virtual time
Driven by three parameters:
- the minimum duration of the sampling interval
-the maximum duration of the sampling interval
-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 secondary memory
demand cleaning - a page is written out to secondary memory only when it has been selected for replacement
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
Process Suspension - If the degree of multiprogramming is to be reduced, one or more of the currently resident processes must be swapped out.
Six possibilities exist:
-lowest-priority process
-Faulting process
-Last process activated
-Process with the smallest resident set
-Largest process
-Process with the largest remaining execution window
UNIX - -intended to be machine so its memory management schemes will vary
early Unix: variable partitioning with no virtual memory scheme
current implementation of UNIX and Solaris make use of paged virtual memory
SVR4 and Solaris use two separate schemes:
-paging system
-kernel memory allocator
Page replacement - -the page frame data table is used for page replacement
-Pointers are used to create lists within the table
-all available frames are linked together in a list of free frames available for bringing in pages
-when the number of available frames drop below a certain threshold, the kernel will steal a number of frames to compensate
Kernel Memory Allocator - the kernel generates and destroys small tables and buffers frequently during the course of execution, each of which requires dynamic memory allocation
-Most of these blocks are significantly smaller than typical pages (therefore paging would be inefficient)
-Allocations and free operations must be made as fast as possible
CHAPTER 9: UNIPROCESSOR SCHEDULING
Processor Scheduling - - aim is to assign processes to be executed by the processor in a way that meets system objectives, such as response time, throughput, and processor efficiency.
Broken down into three sperate functions:
-long term scheduling
-medium term scheduling
-short term scheduling
Types of Scheduling -
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
Medium-term scheduling - -Part of the swapping function
-Swapping-in decisions are based on the need to manage of the degree of multiprogramming
-considers the memory requirements of the swapped out processes
Short term Scheduling - -Known as the dispatcher,
-executes most frequently
-makes the fine grained decision of which process to execute next
-Invoked when an event occurs the 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 (eg. semaphores)
Short term scheduling criteria - -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 processes (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-focus in on effective and efficient utilization of the processor (rate at which processes are completed)
-generally of minor importance on single-user systems
Performance related criteria
-quantitative-easily measured
examples: response time
throughput
Non performance related
-qualitative
-hard to measure
examples: predictability
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:
Nonpreemptive
Preemptive
Nonpreemptive once a process is in the running state, it will continue until it terminates or blocks itself for I/O
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
First-Come-First-Served (FCFS)
Simplest scheduling policy
Also known as first-in-first-out (FIFO) or a strict queuing 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 preempted
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
Shortest Process Next 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 because 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
Performance Comparison
Any scheduling discipline that chooses the next item to be served independent of service time obeys the relationship:
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

Traditional UNIX Scheduling Used in both SVR3 and 4.3 BSD UNIX
these systems are primarily targeted at the time-sharing interactive environment
Designed to provide good response time for interactive users while ensuring that low-priority background jobs do not starve
Employs multilevel feedback using round robin within each of the priority queues
Makes use of one-second preemption
Priority is based on process type and execution history

CHAPTER 10
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
Independent Parallelism No explicit synchronization among processes
each represents a separate, independent application or job
Typical use is in a time-sharing system
each user is performing a particular application
multiprocessor provides the same service as a multiprogrammed uniprocessor
because more than one processor is available, average response time to the users will be
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
The approach taken will depend on the degree of granularity of applications and the number of processors available
Scheduling on a multiprocessor involves three interrelated issues:
• The assignment of processes to processors
• The use of multiprogramming on individual processors
• The actual dispatching of a process
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 o be determined
If a process 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
Assignment of Processes to Processors
Both dynamic and static methods require some way of assigning a process to a processor
Approaches:
Master/Slave
Peer
Master/Slave Architecture
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
Conflict 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
Peer Architecture
Kernel can execute on any processor
Each processor does self-scheduling from the pool of available processes
Complicates the operating system
operating system must ensure that two processors do not choose the same process and that the processes are not somehow lost from the 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
Four approaches for multiprocessor thread scheduling and processor assignment are:
Load Sharing processes are not assigned to a particular processor
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
Dedicated Processor Assignment provides implicit scheduling defined by the assignment of threads to processors
Dynamic Scheduling the number of threads in a process can be altered during the course of execution
Load Sharing Simplest approach and carries over most directly from a uniprocessor environment
Advantages:
load is distributed evenly across the processors
no centralized scheduler required
the global queue can be organized and accessed using any of the schemes discussed in Chapter 9
Versions of load sharing:
first-come-first-served
smallest number of threads first
preemptive smallest number of threads first
Disadvantages of Load Sharing
Central queue occupies a region of memory that must be accessed in a manner that enforces mutual exclusion
can lead to bottlenecks
Preemptive threads are unlikely to resume execution on the same processor
caching can become 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
the process switches involved may seriously compromise performance
Gang Scheduling
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
Dynamic Scheduling
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
this would allow the operating system to adjust the load to improve utilization
Both the operating system and the application are involved in making scheduling decisions
The scheduling responsibility of the operating system 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 operating system, 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
Examples:
control of laboratory experiments
process control in industrial plants
robotics
air traffic control
telecommunications
military command and control systems
Hard real-time task one that must meet its deadline
otherwise it will cause unacceptable damage or a fatal error to the system
Soft real-time task Has an associated deadline that is desirable but not mandatory
It still makes sense to schedule and complete the task even if it has passed its deadline
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
Responsiveness
User control
Reliability
Fail-soft operation
DeterminismConcerned 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
The extent to which an operating system can deterministically satisfy requests depends on:
the speed with which it can respond to interrupts
whether the system has sufficient capacity to handle all requests within the required 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, and data flows external to the system
Concerned with how long, after acknowledgment, it takes an operating system to service the interrupt
Responsiveness includes:
amount of time required to initially handle the interrupt and begin execution of the interrupt service routine (ISR)
amount of time required to perform the ISR
effect of interrupt nesting

User Control
Generally much broader in a real-time operating system than in ordinary operating systems
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
May allow user to specify such characteristics as:
paging or process swapping
what processes must always be resident in main memory
what disk transfer algorithms are to be used
what rights the processes in various priority bands have
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 degradation of performance may have catastrophic consequences such as:
financial loss
major equipment damage
loss of life
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 always met
Real-Time Scheduling
Scheduling approaches depend on:
-whether a system performs schedulability analysis
if it does, whether it is done statically or dynamically
-whether the result of the analysis itself produces a scheduler plan according to which tasks are dispatched at run time
Classes of Real-Time Scheduling Algorithms
Static table-driven approaches
performs a static analysis of feasible schedules of dispatching
result is a schedule that determines, at run time, when a task must begin execution
Static priority-driven preemptive approaches
a static analysis is performed but no schedule is drawn up
analysis is used to assign priorities to tasks so that a traditional priority-driven preemptive scheduler can be used
Dynamic planning-based approaches
feasibility is determined at run time rather than offline prior to the start of execution
one result of the analysis is a schedule or plan that is used to decide when to dispatch this task
Dynamic best effort approaches
no feasibility analysis is performed
system tries to meet all deadlines and aborts any started process whose deadline is missed
Deadline Scheduling
Real-time operating systems are designed with the objective of starting real-time tasks as rapidly 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
Starting deadline
-time task must begin
Completion deadline
-time task must be completed
Processing time

time required to execute the task to completion
Resource requirements
resources required by the task while it is executing
Priority
measures relative importance of the task
Subtask scheduler
a task may be decomposed into a mandatory subtask and an optional subtask
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
CHAPTER 11I/O Management and Disk Scheduling

Categories of I/O Devices
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

Differences in I/O Devices
Devices differ in a number of areas:
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
Organization of the I/O Function
Three techniques for performing I/O are:
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
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
Evolution of the 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
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
Hierarchical Design Functions of the operating system should 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
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
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
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
The 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 Performance Parameters The actual details of disk I/O operation depend on the:
computer system
operating system
nature of the I/O channel and disk controller hardware
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

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
Priority (PRI) Control of the scheduling is outside the control of disk management software
Goal is not to optimize disk utilization but to meet other objectives
Short batch jobs and interactive jobs are given higher priority
Provides good interactive response time
Longer jobs may have to wait an excessively long time
A poor policy for database systems
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
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
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 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 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 5Similar 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 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
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
when an I/O request is made for a particular sector, a check is made to determine if the sector is in the disk cache
if YES the request is satisfied via the cache
if NO the requested sector is read into the disk cache from the disk
Least Recently Used (LRU) Most commonly used algorithm that deals with the design issue of replacement strategy
The block that has been in the cache the longest with no reference to it is replaced
A stack of pointers reference the cache
most recently referenced block is on the top of the stack
when a block is referenced or brought into the cache, it is placed on the top of the stack