Chapter 1

- An ***operating system*** acts as an intermediary between the user of a computer and the computer ***hardware***.
- The purpose of an ***operating system*** is to provide an ***environment*** in which a user can execute programs in a convenient and efficient manner.
- An operating system is ***software*** that ***manages*** the computer hardware.
- An ***operating system*** is a program that ***manages*** the computer hardware. It also provides a basis for application programs and acts as an ***intermediary*** between the computer ***user*** and the computer ***hardware***.
- A computer system can be divided roughly into four components: the ***hardware***, the ***operating system***, the ***application programs***, and the ***users*.**
- The operating system ***controls*** and ***coordinates*** the use of the ***hardware*** among the various application programs for the various users.
- Typically, bootstrap program is stored in read-only memory (***ROM***) or electrically erasable programmable read-only memory (EEPROM), known by the general term ***firmware***, within the computer hardware.
- Hardware may trigger an ***interrupt*** at any time by sending a signal to the CPU, usually by way of the system bus.
- Interrupt transfers control to the interrupt service routine generally, through the ***interrupt vector***, which contains the ***starting addresses*** of all the service routines.
- A ***trap*** is a software-generated interrupt caused either by an error or a user request.
- An operating system is ***interrupt*** driven.
- ***Main memory*** is the only large storage area (millions to billions of bytes) that the processor can access directly.
- Main memory is a ***volatile*** storage device that ***loses*** its contents when power is turned off or otherwise lost.
- The main differences among the various storage systems lie in ***speed***, ***cost***, ***size***, and ***volatility***.
- A general-purpose computer system consists of CPUs and multiple device ***controllers*** that are connected through a ***common bus***.
- Each device ***controller*** is in charge of a specific type of device.
- A device controller maintains some local buffer storage and a set of special-purpose ***registers***.
- The device ***controller*** is responsible for moving the data between the peripheral ***devices*** that it controls and its ***local buffer storage***.
- Typically, operating systems have a device ***driver*** for each device ***controller***.
- Device ***driver*** understands the device ***controller*** and presents a uniform ***interface*** to the device to the rest of the ***operating system.***
- In a parallel system, by increasing the number of processors, we expect to get more work done in less time. The speed-up ratio with N processors is ***less*** than N.
- The difference between ***blade-servers*** and traditional multiprocessor systems is that each blade-processor board boots independently and runs its own operating system.
- Some blade-server boards are multiprocessor as well.

- In essence, those ***blade-servers*** consist of multiple independent multiprocessor systems.
- Clustering can be structured asymmetrically or symmetrically.
- In ***asymmetric*** clustering, one machine is in ***hot-standby*** mode while the other is running the applications. The hot-standby host machine does nothing but ***monitor*** the active server. If that server fails, the hot-standby host becomes the active server.
- In ***symmetric*** mode, two or more hosts are running applications, and are ***monitoring*** each other.
- A ***time-shared*** operating system allows many users to share the computer simultaneously. As the system switches rapidly from one user to the next, each user is given the impression that the entire computer system is dedicated to his use, even though it is being shared among many users.
- A time-shared operating system uses CPU ***scheduling*** and ***multiprogramming*** to provide each user with a small portion of a time-shared computer.
- If several jobs are ready to be brought into memory, and if there is not enough room for all of them, then the system must choose among them. Making this decision is ***job scheduling***.
- if several jobs are ready to run at the same time, the system must choose among them. Making this decision is ***CPU scheduling.***
- ***Virtual memory*** is a technique that allows the execution of a process that is not completely in memory.
- The main advantage of the virtual-memory scheme is that it enables users to run programs that are ***larger*** than actual physical ***memory***.
- A ***trap*** (or an ***exception***) is a software-generated interrupt caused either by an error or by a specific request from a user program that an operating-system service be performed.
- In an operating system, at least, we need two separate modes of operation: ***user*** mode and ***kernel*** mode (also called ***supervisor*** mode, ***system*** mode, or ***privileged*** mode).
- At system boot time, the hardware starts in ***kernel*** mode. The operating system is then loaded and starts user applications in ***user*** mode. Whenever a trap or interrupt occurs, the hardware switches from user mode to kernel mode (that is, changes the state of the mode bit to 0). Thus, whenever the operating system gains control of the computer, it is in ***kernel*** mode. The system always switches to ***user*** mode (by setting the mode bit to 1) before passing control to a user program.
- The hardware allows ***privileged*** instructions to be executed only in kernel mode.
- If an attempt is made to execute a ***privileged*** instruction in user mode, the hardware does not execute the instruction but rather treats it as illegal and traps it to the ***operating system***.
- ***System calls*** provide the means for a user program to ask the operating system to perform tasks reserved for the operating system on the user program's behalf.
- A ***process*** needs certain resources-including CPU time, ***memory***, files, and ***I/O devices*** - to accomplish its task. These resources are either given to the process when it is created or allocated to it while it is running.

- A ***process*** is the unit of work in a system.
- The ***main memory*** is generally the only large storage device that the CPU is able to address and access directly.
- Files are normally organized into ***directories*** to make them easier to use.
- Most modern computer systems use ***disks*** as the principal on-line storage medium for both programs and data.
- Only the ***device driver*** knows the peculiarities of the specific device to which it is assigned.
- A ***distributed*** system is a collection of physically ***separate***, possibly heterogeneous computer systems that are networked to provide the users with access to the various ***resources*** that the system maintains.
- Embedded systems almost always run ***real-time operating*** systems.
- A ***real-time*** system has well-defined, fixed ***time constraints.*** Processing *must* be done ***within*** the defined constraints, or the system will fail.

Chapter 2

- ***OSes*** provide an environment for execution of programs and services to programs and users.
- Communications may be implemented via ***shared memory*** or through ***message passing***, in which packets of information are moved between processes by the operating system.
- The main function of the command ***interpreter*** is to get and ***execute*** the next user-specified command.
- There are two general ways in which these commands can be implemented. In one approach, the command interpreter itself ***contains*** the code to execute the command. An alternative implements most commands through system ***programs***.
- Three general methods are used to pass parameters to the operating system.
    1. The simplest approach is to pass the parameters in ***registers***.
    2. In some cases, there may be more parameters than registers. In these cases, the parameters are generally stored in a ***block***, or ***table***, in memory, and the ***address*** of the block is passed as a parameter in a register. This is the approach taken by Linux and Solaris.
    3. Parameters also can be placed, or *pushed*, onto the ***stack*** by the program and ***popped*** off the stack by the operating system.
- System calls can be grouped roughly into the following major categories: ***process*** control, file manipulation, device manipulation, information maintenance, communications and protections.
- ***Debugger*** is a system program designed to aid the programmer in finding and correcting bugs.
- There are two common models of interprocess communication: the ***message passing*** model and the ***shared-memory*** model.
- ***System*** programs provide a convenient environment for program development and

execution.

- One important principle is the separation of policy from mechanism. ***Mechanisms*** determine *how* to do something; ***policies*** determine *what* will be done.
- ***System calls*** allow a running program to make requests from the operating system directly.
    1. The four typically structures for OS are simple structure, ***layered***, ***microkernels*** and ***modules***.
    2. Microkernel contains only essential OSes functions, including ***memory management***, ***CPU scheduling,*** and ***IPC***.
- **A *virtual machine*** treats hardware and the operating system kernel as though they were all hardware.
- ***Virtualization*** is a technology that allows operating systems to run as applications within other operating systems.
- JVM consists of ***class loader,*** class verifier, and Java ***interpreter*.**
- The procedure of starting a computer by loading the kernel is known as ***booting*** the system.
- On most computer systems, a small piece of code known as the ***bootstrap program*** or ***bootstrap loader*** locates the kernel, loads it into main memory, and starts its execution.

Chapter 3

- A ***process*** can be thought of as a program in execution.
- A ***process*** is a program in execution.
- A ***process*** is the unit of work in a modern time-sharing system.
- A process will need certain resources-such as ***CPU time, memory, files,*** and ***I/O devices*** -to accomplish its task. These resources are allocated to the process either when it is ***created*** or while it is ***executing***.
- A program becomes a ***process*** when an executable file is loaded into ***memory***.
- Each process may be in one of the following states: new, ***running, waiting, ready***, and terminated.
- Only one process can be ***running*** on any processor at any instant. Many processes may be ***ready*** and ***waiting***.
- Each process is represented in the operating system by a ***process control block (PCB).***
- The objective of multiprogramming is to have some process ***running*** at all times, to ***maximize*** CPU utilization.
- The objective of ***time sharing*** is to switch the ***CPU*** among processes so frequently that users can interact with each program while it is ***running***.
- The processes that are residing in main memory and are ready and waiting to ***execute*** are kept on a list called the ***ready*** queue.
- ***CPU scheduler*** selects from among the processes that are ***ready*** to execute and allocates the CPU to one of them.
- The ***long-term*** scheduler controls the degree of multiprogramming (the number of processes in memory).
- The primary distinction between the job scheduler and CPU scheduler lies in ***frequency of execution.***
- The process is swapped out, and is later swapped in, by the ***medium-term*** scheduler.
- The context is represented in the ***PCB*** of the process**.**
- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process. This task is known as a ***context switch*****.**
- The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the ***ready*** queue.
- The list of processes waiting for a particular I/O device is called a ***device*** queue.
- Most operating systems (including UNIX and the Windows family of operating systems) identify processes according to a unique ***process identifier*** (or ***pid***), which is typically an ***integer*** number.
- Cooperating processes require an interprocess communication (IPC) mechanism that will allow them to exchange data and information. There are two fundamental models of interprocess communication: (1) ***shared memory*** and (2) ***message passing***.
- Shared memory is ***faster*** than message passing.
  Because message-passing systems are typically implemented using ***system calls*** and

thus require the more timeconsuming task of kernel intervention.
In shared-memory systems, system calls are required only to establish ***shared-memory regions.*** Once shared memory is established, all accesses are treated as routine memory accesses, and no assistance from the kernel is required.

- Under ***direct communication***, each process that wants to communicate must explicitly name the recipient or sender of the communication.
- With ***indirect*** communication, the messages are sent to and received from ***mailboxes***, or ports.
- Communication in client-server systems may use (1) ***sockets***, (2) remote procedure calls (***RPCs***), or (3) Java's remote method invocation (***RMI***).

Chapter 4

- A ***thread*** is a basic unit of CPU utilization; it comprises a thread ID, a ***program counter,*** a ***register set,*** and a stack. It shares with other threads ***belonging to*** the same process its ***code section***, ***data section,*** and other operating-system ***resources***, such as ***open files*** and signals.
- Threads share the ***memory*** and the ***resources*** of the process to which they belong.
- it is much more ***time consuming*** to create and manage processes than ***threads***.
- Support for threads may be provided either at the ***user*** level, for user threads, or by the ***kernel***, for kernel threads.
- ***User*** threads are supported ***above*** the kernel and are managed ***without*** kernel support, whereas ***kernel*** threads are supported and managed directly by the ***operating system***.
- The many-to-one model maps many user-level threads to one ***kernel*** thread.
- The ***one-to-one*** model maps each user thread to a kernel thread.
- The ***one-to-one*** model provides more concurrency than the many-to-one model by allowing another thread to run when a thread makes a blocking system call.
- The ***one-to-one*** model allows multiple threads to run in parallel on multiprocessors.
- The ***one-to-one*** model, creating a user thread requires creating the corresponding kernel thread.
- The ***many-to-many*** model multiplexes many user-level threads to a smaller or equal number of kernel threads.
- The ***many-to-one*** model allows the developer to create as many user threads as she wishes, true concurrency is not gained because the kernel can schedule only one thread at a time.
- A ***thread*** is a flow of control within a process.
- A multithreaded process contains several different flows of ***control*** within the same address space.
- The benefits of multithreading include increased responsiveness to the user, resource sharing within the process, economy, and the ability to take advantage of ***multiprocessor*** architectures.
- ***User-level*** threads are threads that are visible to the programmer and are unknown to

the kernel.

- The operating-system kernel supports and manages ***kernel-level*** threads.
- Three different types of models relate user and kernel threads: The ***many-to-one*** model maps many user threads to a single kernel thread. The ***one-to-one*** model maps each user thread to a corresponding kernel thread. The ***many-to-many*** model multiplexes many user threads to a smaller or equal number of kernel threads.
- ***Thread libraries*** provide the application programmer with an API for creating and managing threads. Three primary thread libraries are in common use: ***POSIX Pthreads***, ***Win32*** threads for Windows systems, and ***Java*** threads.
- A ***signal*** is used in UNIX systems to notify a process that a particular event has occurred.
- A signal may be received either synchronously or asynchronously, depending on the source of and the ***reason*** for the event being signaled.
- ***Synchronous*** signals are delivered to the same process that performed the operation that ***caused*** the signal.
- When a signal is generated by an event ***external*** to a ***running*** process. That process receives the signal asynchronously.
- ***Synchronous*** signals need to be delivered to the thread ***causing*** the signal and not to other threads in the process.
- To the user-thread library, the ***LWP*** appears to be a *virtual processor* on which the application can schedule a user thread to run. Each LWP is attached to a kernel thread, and it is ***kernel threads*** that the operating system schedules to run on physical processors.
- In general, user-level threads are ***faster*** to create and manage than are kernel threads, as no intervention from the kernel is required.

Chapter 5

- In a single-processor system, only ***one*** process can run at a time; any others must wait until the CPU is free and can be rescheduled.
- The objective of multiprogramming is to have some process ***running*** at all times, to maximize ***CPU*** utilization.
- Process execution consists of a cycle of ***CPU*** execution and ***I/O*** wait. Processes alternate between these two states. Process execution ***begins*** with a ***CPU burst.*** That is ***followed*** by an ***I/O burst***, which is followed by another ***CPU burst,*** then another ***I/O burst,*** and so on. Eventually, the ***final CPU burst*** ends with a system request to terminate execution.
- An ***I/O-bound*** program typically has many ***short*** CPU bursts. A ***CPU-bound*** program might have a few ***long*** CPU bursts.
- Whenever the CPU becomes ***idle***, the operating system must select one of the processes in the ***ready*** queue to be executed. The selection process is carried out by the ***short-term*** scheduler (or ***CPU*** scheduler).
- The ***scheduler*** selects a process from the processes in memory that are ***ready*** to execute and allocates the CPU to that process.

- All the processes in the ready queue are lined up waiting for a chance to ***run*** on the CPU. The records in the queues are generally ***process control blocks (PCBs)*** of the processes.
- Under ***nonpreemptive*** scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the ***waiting*** state.
- The ***dispatcher*** is the module that gives control of the ***CPU*** to the process selected by the short-term scheduler.
- The time it takes for the ***dispatcher*** to stop one process and start another running is known as the dispatch ***latency***.
- The interval from the time of submission of a process to the time of completion is the ***turnaround time***.
- ***Waiting time*** is the sum of the periods spent waiting in the ready queue.
- The CPU scheduling algorithm does not affect the amount of time during which a process executes or does I/O; it affects only the amount of time that a process spends ***waiting*** in the ***ready*** queue.
- ***Response time*** is the time from the submission of a request until the first response is produced.
- ***Response time*** is the time it takes to start responding, not the time it takes to output the response.
- The PCPS scheduling algorithm is ***nonpreemptive***.
- The SJF algorithm can be either ***preemptive*** or ***nonpreemptive***.
- Priority scheduling can be either ***preemptive*** or ***nonpreemptive***.
- The RR scheduling algorithm is thus ***preemptive***.
- Only ***nonpreemptive*** algorithms can be used for job scheduling.
- ***Preemptive*** and ***nonpreemptive*** algorithms can be used for CPU scheduling.

Chapter 6

- A situation like this, where several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place, is called a ***race condition.***
- A ***critical section*** is a segment of ***code***, in which the process may be changing common variables, updating a table, writing a file, and so on.
- The ***critical-section problem*** is to design a ***protocol*** that the processes can use to cooperate.
- when one process is executing in its critical section, no other process is to be ***allowed*** to execute in its critical section.
- A solution to the critical-section problem must satisfy the following three requirements: ***mutual exclusion, progress,*** and ***bounded waiting***.
- A semaphore S is an integer variable that, apart from initialization, is accessed only through two standard atomic operations: ***wait( )*** and ***signal( ).***
- All the modifications to the integer value of the semaphore in the wait() and signal() operations must be executed ***indivisibly***.

- Operating systems often distinguish between counting and binary semaphores. The value of a ***counting*** semaphore can range over an unrestricted domain. The value of a ***binary*** semaphore can range only between 0 and 1.
- Counting semaphores can be used to control access to a given resource consisting of a finite number of instances. The semaphore is initialized to the ***number*** of resources available.
- Each process that wishes to use a resource performs a ***wait()*** operation on the semaphore. When a process releases a resource, it performs a ***signal()*** operation.
- When the count for the semaphore goes to ***0***, all resources are being used.
- A process that is blocked, waiting on a semaphore S, should be restarted when some ***other*** process executes a ***signal()*** operation.
- The ***block()*** operation suspends the process that invokes it. The ***wakeup(P)*** operation resumes the execution of a blocked process P.
- If the semaphore value is negative, its magnitude is the number of processes ***waiting on*** that semaphore.
- We must guarantee that no two processes can execute wait() and signal() operations on the ***same*** semaphore at the same time.
- A set of processes is in a ***deadlock*** state when every process in the set is waiting for an event that can be caused only by another process in the set.
- ***Starvation*** or indefinite ***blocking*** is a situation in which processes wait indefinitely within the semaphore.
- The ***monitor*** type is one fundamental high-level synchronization construct.
- The ***monitor*** type contains the declaration of variables whose values define the state of an instance of that type, along with the bodies of procedures or functions that operate on those variables.
- A procedure defined within a monitor can ***access only*** those variables declared locally within the monitor and its formal parameters. Similarly, the local variables of a monitor can ***be accessed by only*** the local procedures.

Chapter 7

- A set of processes is in a ***deadlock*** state when every process in the set is waiting for an event that can be caused only by another process in the set.
- A ***deadlock*** situation can arise if the following four conditions hold simultaneously in a system: ***mutual exclusion, hold and wait, no preemption,*** and ***circular wait.***
- If the resource-allocation graph contains no cycles, then no process in the system is ***deadlocked***.
- In a given resource-allocation graph,
    1. if each resource type has ***exactly one*** instance, then a ***cycle*** implies that a ***deadlock*** has occurred.
    2. if the cycle involves only a set of resource types, each of which has only a ***single*** instance, then a deadlock has occurred.
    3. Each process involved in the cycle is ***deadlocked***.
    4. In this case, a cycle in the graph is both a ***necessary*** and a ***sufficient***

condition for the existence of deadlock.

- If each resource type has several instances, then a cycle does not necessarily imply that a deadlock has occurred. In this case, a cycle in the graph is a ***necessary*** but not a ***sufficient*** condition for the existence of deadlock.
- We can use a protocol to ***prevent*** or ***avoid*** deadlocks. ensuring that the system will ***never enter*** a deadlock state.
- To ensure that deadlocks never occur, the system can use either a ***deadlock-prevention*** or a ***deadlock-avoidance*** scheme.
- ***Deadlock prevention*** provides a set of methods for ensuring that at least one of the ***necessary*** conditions cannot hold.
- For a deadlock to occur, each of the four necessary conditions must hold. By ensuring that at least one of these conditions cannot hold, we can ***prevent*** the occurrence of a deadlock.
- The mutual-exclusion condition must hold for ***nonsharable*** resources.
- ***Sharable*** resources (such as read-only files) do not require mutually exclusive access and thus cannot be involved in a deadlock.
- To ensure that the ***hold-and-wait*** condition never occurs in the system, we must guarantee that, whenever a process requests a resource, it does not hold any other resources.
- The ***deadlock-avoidance*** algorithm dynamically examines the resource-allocation state to ensure that a ***circular-wait*** condition can never exist.
- A state is ***safe*** if the system can allocate resources to each process (up to its maximum) in some order and still avoid a deadlock.
- A system is in a ***safe*** state only if there exists a ***safe sequence.***
- If no such sequence exists, then the system state is said to be ***unsafc***.
- A ***deadlock*** state occurs when two or more processes are waiting indefinitely for an event that can be caused only by one of the waiting processes.
- A deadlock can occur only if four necessary conditions hold simultaneously in the system: ***mutual exclusion, hold and wait, no preemption***, and ***circular wait.***
- To ***prevent*** deadlocks, we can ensure that at least one of the necessary conditions never holds.

Chapter 8

- The main purpose of a computer system is to execute programs. These programs, together with the data they access, must be in ***main memory*** (at least partially) during execution.
- The CPU fetches instructions from memory according to the value of the ***program counter.***
- A typical instruction-execution cycle, first fetches an instruction from ***memory***. The instruction is then ***decoded*** and may cause operands to be ***fetched*** from ***memory***. After the instruction has been ***executed*** on the operands, results may be ***stored*** back in ***memory***.
- The base and limit registers can be loaded only by the operating system, which uses a special ***privileged*** instruction.
- ***Privileged*** instructions can be executed only in kernel mode, and since only the operating system executes in kernel mode, only the operating system can load the memory base and limit registers.
- An address generated by the CPU is commonly referred to as a ***logical*** address.
- The address loaded into the memory-address register of the memory is commonly referred to as a ***physical*** address,
- In ***compile-time*** and ***load-time*** address-binding schemes, logical and physical addresses are the ***same***.
- In ***execution-time*** address-binding scheme, logical (virtual) and physical addresses ***differ***.
- We can place the operating system in either low memory or high memory. The major factor affecting this decision is the location of the ***interrupt vector.***
- In the contiguous memory allocation, each process is contained in a ***single contiguous*** section of memory.
- The memory allocated to a process may be slightly larger than the requested memory. The difference between these two numbers is ***internal fragmentation***.
- ***Internal fragmentation*** refers to the memory that is internal to a partition but is not being used.
- ***External fragmentation*** exists when there is enough total memory space to satisfy a request, but the available spaces are not contiguous; storage is fragmented into a large number of small holes.
- A 32-bit page-table entry can point to one of 2^{32} physical page frames. If frame size is 4 KB, then a system with 4-byte entries can address 2^{44} bytes (or ***16 TB***) of physical memory.
- Instructions to load or modify the page-table registers are ***privileged***, so that only the operating system can change the memory map.
- ***Segmentation*** is a memory-management scheme that supports this user view of memory.
- In ***segmentation*** scheme, a logical address consists of two parts: a ***segment number***, *s*, and an ***offset*** into that segment, *d*. The segment number is used as an index to the ***segment table.*** The offset *d* of the logical address must be between 0 and the ***segment limit.***
- In segmentation scheme, the user specifies each address by two quantities: a segment name and an offset. Contrast this scheme with the ***paging*** scheme, in which the user specifies only a single address, which is ***partitioned*** by the ***hardware*** into a page number and an offset, all invisible to the programmer.
- Systems with fixed-sized allocation units, such as the single-partition scheme and paging, suffer from ***internal*** fragmentation.

- Systems with variable-sized allocation units, such as the multiple-partition scheme and segmentation, suffer from ***external*** fragmentation.

Chapter 9

- ***Virtual*** memory is a technique that allows the execution of processes that are not completely in memory.
- Virtual address spaces that include holes are known as ***sparse*** address spaces.
- In addition to separating logical memory from physical memory, virtual memory also allows files and memory to be ***shared*** by two or more processes through page sharing.
- Consider how an executable program might be loaded from disk into memory.
    1. One option is to load the entire program in physical memory at program execution time.
    2. An alternative strategy is to initially load pages only as they are needed.
       This technique is known as ***demand paging*** and is commonly used in virtual memory systems.
- Virtual memory is commonly implemented by ***demand paging***.
- A demand-paging system is similar to a paging system with swapping where processes reside in secondary memory. When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory, however, we use a ***lazy swapper.***
- lazy swapper ***never*** swaps a page into memory unless that page will be needed.
- Since we are now viewing a process as a sequence of ***pages***, rather than as one large ***contiguous*** address space.
- A ***swapper*** manipulates ***entire*** processes, whereas a ***pager*** is concerned with the individual ***pages*** of a process.
- What happens if the process tries to access a page that was not brought into memory?
    1. Access to a page marked invalid causes a ***page-fault trap***.
    2. The paging hardware, in ***translating*** the address through the ***page table***, will notice that the ***invalid*** bit is ***set,*** causing a trap to the operating system.
- The ***effective access time*** is directly ***proportional*** to the ***page-fault rate***.
- It is important to keep the page-fault rate ***low*** in a demand-paging system. Otherwise, the effective access time increases, ***slowing*** process execution dramatically.
- Disk I/O to ***swap*** space is generally ***faster*** than that to the ***file system***, because swap space is allocated in much larger blocks, and file lookups and indirect allocation methods are not used .
- The system can therefore gain better paging throughput by
    1. copying an entire file image into the swap space at process ***startup*** and then performing demand paging from the swap space.
    2. to read demand pages from the file system initially but to write the pages to swap space as they are replaced. This approach will ensure that only ***needed*** pages are read from the file system but that all subsequent paging is done from swap space.
- We evaluate an algorithm by running it on a particular string of memory references and computing the number of page faults. The string of memory references is called a ***reference*** string.
- The key distinction between the FIFO and OPT algorithms is that the FIFO algorithm uses the time when a page was ***brought into*** memory, whereas the OPT algorithm uses the time when a

page is to be ***used***.

- If we use the recent past as an approximation of the near future, then we can replace the page that ***has not been used*** for the ***longest*** period of time. This approach is the ***least-recently-used (LRU)*** algorithm**.**
- The ***reference*** bit for a page is set by the ***hardware*** whenever that page is referenced (either a read or a write to any byte in the page).
- Reference bits are associated with each ***entry*** in the page table.
- Second-chance replacement degenerates to ***FIFO*** replacement if all bits are set.
- This high paging activity is called ***thrashing***.
- A process is ***thrashing*** if it is spending more time paging than executing.
- A ***locality*** is a set of pages that are actively used together, which is defined by the program structure and its data structures.
- If the total demand for frames is greater than the total number of available frames, ***thrashing*** will occur, because some processes will not have enough frames.
- the ***working-set model*** is based on the assumption of ***locality***，which are defined by the program structure and its data structures.
- Thrashing has a ***high*** page-fault rate.
- If a process does not have enough memory for its working set, it will ***thrash***.
- ***Memory mapping*** a file allows a part of the virtual address space to be logically associated with the file.
- Memory mapping a file is accomplished by mapping a disk block to a ***page*** in memory.
- To provide memory-mapped I/O, ranges of memory addresses are set aside and are mapped to the device ***registers*****,** called an I/O port**.**
- The ***buddy*** system allocates memory from a fixed-size segment consisting of physically contiguous pages by using a ***power-of-2*** allocator, which satisfies requests in units sized as a power of 2.
- For ***slab*** allocation,
    1. A slab is made up of one or more physically contiguous pages.
    2. A ***cache*** consists of one or more ***slabs***.
    3. There is a single cache for each unique kernel data structure.
- The buddy system allocates memory to kernel processes in units sized according to a power of ***2***, which often results in ***fragmentation***.
- Slab allocators assign kernel data structures to caches associated with slabs, which are made up of one or more physically ***contiguous*** pages. With slab allocation, no memory is wasted due to ***fragmentation***,
- To minimize internal fragmentation, we need a ***small*** page size.
- A desire to minimize I/O time argues for a ***larger*** page size.
- A ***smaller*** page size allows each page to match program locality more accurately.
- To minimize the number of page faults, we need to have a ***large*** page size.
- The TLB reach refers to the amount of memory accessible from the TLB and is simply the number of entries multiplied by the ***page size***.

Chapter 10

- Modern computer systems use ***disks*** as the primary on-line storage medium for information (both programs and data).
- The ***file system*** provides the mechanism for on-line storage of and access to both data and programs residing on the disks,
- A ***file*** is a collection of related information defined by its ***creator***.
- Files are normally organized into ***directories*** for ease of use.
- Because of all this device variation，one key goal of an operating system's I/O subsystem is to provide the simplest ***interface*** possible to the rest of the system.
- The file system is the most visible aspect of an operating system. It provides the mechanism for on-line ***storage*** of and ***access*** to both data and programs of the operating system.
- The file system consists of two distinct parts: a collection of ***files***, each storing related data, and a ***directory structure***, which organizes and provides information about all the files in the system.
- File ***name*** is the only information kept in ***human-readable*** form, and file ***identifier*** is the ***unique*** tag, usually a number, identifies the file within the file system, it is the ***non-human-readable*** name for the file.
- The minimal set of required file operations includs ***create, write, read, reposition, delete,*** and ***truncate***.
- A common technique for implementing file types is to include the type as part of the file name. The name is split into two parts-a ***name*** and an ***extension***, usually separated by a period character.
- Application programs also use ***extensions*** to indicate file ***types*** in which they are interested.
- Disk systems typically have a well-defined block size determined by the size of a ***sector***.
- All disk I/O is performed in units of one ***block*** (physical record), and all blocks are the ***same*** size.
- The UNIX operating system defines all files to be simply streams of ***bytes***. Each byte is individually addressable by its ***offset*** from the beginning (or end) of the file. In this case, the ***logical*** record size is 1 byte.
- The ***logical*** record size, ***physical*** block size, and ***packing*** technique determine how many logical records are in each physical block.
- The basic I/O functions operate in terms of ***blocks***.
- Because disk space is always allocated in ***blocks***, some portion of the last block of each file is generally wasted. The waste incurred to keep everything in units of blocks (instead of bytes) is ***internal*** fragmentation.
- All file systems suffer from ***internal*** fragmentation; the ***larger*** the block size, the ***greater*** the ***internal*** fragmentation.
- ***Sequential*** access is based on a tape model of a file and works as well on sequential-access ***devices*** as it does on random-access ones.
- The ***direct-access*** method is based on a ***disk*** model of a file, since disks allow random access to any file block.
- There are no restrictions on the ***order*** of reading or writing for a direct-access file.
- The block number provided by the user to the operating system is normally a ***relative*** block

number. A relative block number is an index relative to the ***beginning*** of the file. The first relative block of the file is ***0***.

Chapter 11

- The file system provides the mechanism for on-line ***storage*** and access to file contents, including ***data*** and programs.
- The file system resides permanently on ***secondary storage***, which is designed to hold a large amount of data permanently.
- To improve l/O efficiency, I/O transfers between memory and disk are performed in units of ***blocks***.
- A file system poses two quite different design problems. The first problem is defining the file system interface. This task involves defining a file and its attributes, the operations allowed on a file, and the directory structure for organizing files. The second problem is creating algorithms and data structures to ***map*** the logical file system onto the physical secondary-storage devices.
- The lowest level, the ***I/O control,*** consists of ***device drivers*** and ***interrupt handlers*** to transfer information between the main memory and the disk system.
- The device ***driver*** usually writes specific bit patterns to special locations in the I/O ***controller's*** memory to tell the ***controller*** which device location to act on and what actions to take.
- The ***basic file system*** needs only to issue generic commands to the appropriate device ***driver*** to read and write physical blocks on the disk.
- The ***file-organization*** module can translate logical block addresses to physical block addresses for the basic file system to transfer.
- Each file's logical blocks are numbered from 0 (or 1) through N.
- The ***file-organization*** module also includes the ***free-space*** manager, which tracks unallocated blocks and provides these blocks to the file-organization module when requested.
- The ***logical*** file system manages ***metadata*** information. Metadata includes all of the file-system structure except the actual *data* (or contents of the files).
- The ***logical*** file system manages the ***directory*** structure to provide the file organization module with the information the latter needs, given a symbolic file name. It maintains file structure via file-control blocks.
- A ***file-control*** block (FCB) contains information about the file, including ownership, permissions, and location of the file contents.
- The ***logical*** file system is also responsible for protection and security,
- On-disk, the file system may contain information about how to boot an ***operating*** system stored there, the total number of blocks, the ***number*** and ***location*** of free blocks, the ***directory*** structure, and individual ***files*****.**
- A ***boot control*** block can contain information needed by the system to boot an ***operating system*** from that volume.
- A ***volume control*** block contains volume (or partition) details, such as the number of blocks in the partition, size of the blocks, free block count and free-block pointers, and free FCB count and FCB pointers.
- A per-file ***FCB*** contains many details about the file, including file permissions, ownership, size, and location of the data blocks.

- Three major methods of allocating disk space are in wide use: ***contiguous***, ***linked***, and ***indexed***.
- ***Contiguous*** allocation requires that each file occupy a set of contiguous blocks on the disk.
- When head movement is needed (from the last sector of one cylinder to the first sector of the next cylinder), the head need only move from one track to the next. Thus, the number of disk seeks required for accessing ***contiguously*** allocated files is ***minimal***.
- Both ***sequential*** and ***direct*** access can be supported by ***contiguous*** allocation.
- There is no external fragmentation with ***linked*** allocation.
- It can be used effectively only for ***sequential-access*** files. it is inefficient to support a ***direct-access*** capability for ***linked-allocation*** files.
- ***Indexed*** allocation supports ***direct*** access, without suffering from ***external*** fragmentation.

Chapter 12

- Disk speed has two parts. The ***transfer rate*** is the rate at which data flow between the drive and the computer. The ***positioning time,*** sometimes called the random-access time, consists of the time to move the disk arm to the desired cylinder, called the ***seek*** time, and the time for the desired sector to rotate to the disk head, called the ***rotational latency.***
- The ***disk access time*** has two major. The ***seek*** time is the time for the disk arm to move the heads to the cylinder containing the desired sector. The ***rotational latency*** is the additional time for the disk to rotate the desired sector to the disk head.
- The ***disk bandwidth*** is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.
- ***Low-level formatting*** fills the disk with a special data structure for each sector. The data structure for a sector typically consists of a header, a data area (usually 512 bytes in size), and a trailer.
- The main goal for the design and implementation of swap space is to provide the best ***throughput*** for the virtual memory system.

Chapter 13

- The two main jobs of a computer are I/O and ***processing***.
- The role of the operating system in computer I/O is to manage and control I/O ***operations*** and I/O ***devices***.
- The ***device drivers*** present a uniform ***device access interface*** to the I/O subsystem, much as system calls provide a standard interface between the application and the operating system.
- An I/O port typically consists of four registers, called the (1) ***status***, (2) ***control***, (3) ***data-in,*** and (4) ***data-out*** registers.
- The processor communicates with the controller using special I/O ***instructions*** or ***memory-mapped*** I/O.
- The ***special I/O instructions*** specify the transfer of a byte or word to an I/O port address. The I/O instruction triggers bus lines to select the proper device and to move bits into or out of a device register.
- The ***memory-mapped*** I/O, the device-control registers are mapped into the address space of the processor. The CPU executes I/O requests using the ***standard*** data-transfer instructions to read and write the device-control registers.

- The CPU performs a state save and jumps to the ***interrupt handler*** routine at a fixed address in memory.
- The ***interrupt handler*** determines the ***cause*** of the interrupt, performs the necessary ***processing***, performs a state ***restore***, and executes a ***return*** from interrupt instruction to return the CPU to the execution state prior to the interrupt.
- The device ***controller raises*** an interrupt by asserting a signal on the ***interrupt request line***, the ***CPU catches*** the interrupt and ***dispatches*** it to the interrupt ***handler***, and the ***handler clears*** the interrupt by ***servicing*** the device.
- Most CPUs have two interrupt request lines: the ***nonmaskable*** interrupt line, and the ***maskable*** interrupt line.
- The ***nonmaskable*** interrupt line is reserved for events such as unrecoverable memory errors.
- The ***maskable*** interrupt line can be turned off by the CPU before the execution of critical instruction sequences that must not be interrupted.
- The ***maskable*** interrupt is used by device controllers to request service.
- The purpose of the ***device-driver*** layer is to hide the differences among device ***controllers*** from the I/O subsystem of the kernel, much as the I/O system calls encapsulate the behavior of devices in a few generic classes that hide hardware differences from applications.
- The difference between ***nonblocking*** and ***asynchronous*** system calls is that a nonblocking read () returns ***immediately*** with whatever data are available--the full number of bytes requested, fewer, or none at all. An asynchronous readO call requests a transfer that will be performed in its entirety but that will complete at some ***future*** time.
- The kernel's ***I/O subsystem*** provides several services, including ***scheduling***, ***buffering***, ***caching***, ***spooling***, device ***reservation***, and ***error*** handling.
- A ***buffer*** is a memory area that stores data while they are transferred between two devices or between a device and an application.
- A ***cache*** is a region of fast memory that holds copies of data.
- A ***spool*** is a buffer that holds output for a device, such as a printer, that cannot accept interleaved data streams.