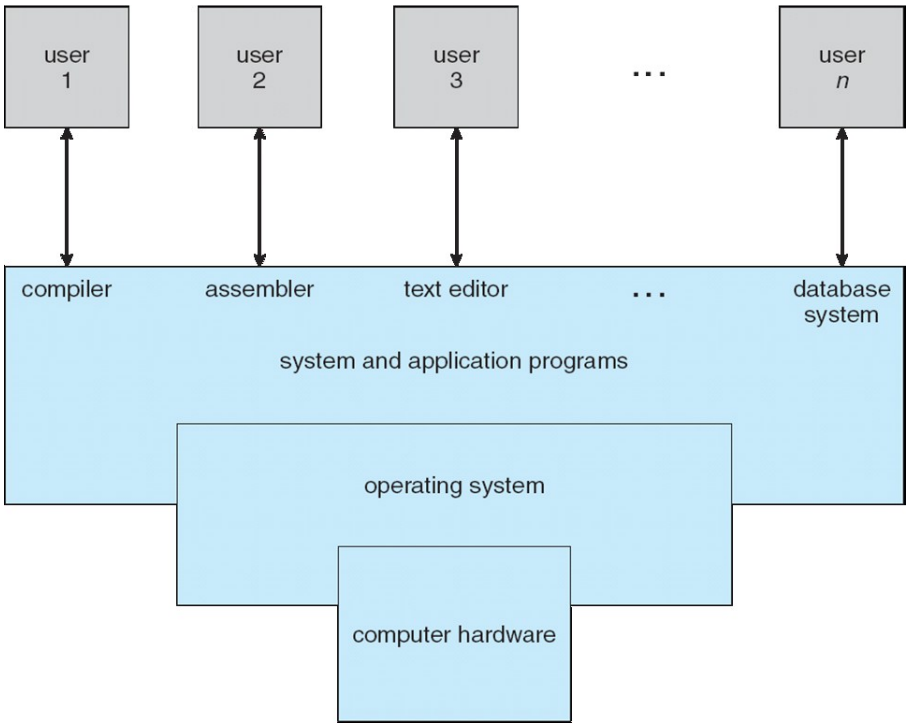
Chapter 1

Know what is an operating system.

An operating system is a program that manages the computer hardware. I provides a basis for application programs and acts as an intermediary between the computer user and the computer hardware. An operating system can also be thought as resource allocator, managing all system resources (I/O, memory, etc) and deciding what to do with requests, especially conflicting requests, to use those resources. The operating system is also called the "Kernel" meaning "This program is running directly on the hardware."

Be familiar with the figure on the slide 1.6



Know the steps to start up a computer.

The bootstrap program is loaded at power-up or reboot. It is typically stored in ROM (Read-Only Memory) or EPROM (Erasable Programmable Read-Only Memory), these two generally being known as firmware. This program initializes all aspects of the system, including loading the operating system into kernel and starting execution. The operating system then starts executing the first process, such as "init," and waits for some event to occur.

Understand the concept of interrupt: What is an interrupt, What does it do, Interrupt handling.

What is it? : An interrupt is a scenario where a device controller informs the CPU it has finished its operation. More specifically, it is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. Software may trigger an interrupt by executing a system call.

What does it do? : An interrupt alerts the processor to a high-priority condition requiring the

interruption of the current code the processor is executing. When the CPU is interrupted, it stops what it is doing and immediately transfers execution to fixed location to resume later. Interrupt transfers control to the interrupt service routine generally through the interrupt vector, which contains the addresses of all service routines.

Interrupt handling : An operating system is interrupt driven, and thus, must have interrupt handling. In which, the operating system preserves the state of the CPU by storing registers and the program counter. It determines what type of interrupt occurred via polling (sampling device status) and the interrupt vector. Different code segments determine the action to take for each type of interrupt.

Know storage hierarchy: Caching, Device driver, DMA,…

General Storage structure:

Main memory – only large storage media that the CPU can access directly (RAM) (Random access and volatile)
Secondary storage – extension of main memory that provides large nonvolatile storage capacity (HHD or SSD)

Storage hierarchy is organized by speed, cost and volatility. Caching is copying information into a faster storage system (main memory) for ease of access and frequent access. The Device driver for each device manages it's I/O via controller. DMA or direct mapped access connects memory to an I/O device directly, sometimes allowing use of the memory without causing an interrupt.

Know computer system architecture: Asymmetric Multiprocessing, Symmetric Multiprocessing, Multicore, Cluster

Asymmetric Multiprocessing – each processor in a multiprocessor system is assigned a specific task

Symmetric Multiprocessing – each processor performs all the tasks in the multiprocessor system (ie they shared the same group of tasks)

A Multicore is a chip containing many processor chips each with their own cache and registers but all connected to the same memory. Similarly, Clustered Systems are multiple systems working together rather than just one system. They typically share storage via a storage-area network (SAN) and, like a multiprocessor system, can be Asymmetric or Symmetric is responding to jobs. Some clusters even uses parallelization (applications written to use multicore processors or systems) to achieve high-performance computing (HPC)

Chapter 2

Be able to describe the services an operating system provides: User interface, Program execution, I/O operations, File-system manipulation, Communications, Error detection, Resource allocation, Accounting, Protection and security. Figure on the slide 2.7

User interface – varies between OS, but is generally an interface to allow a user to use the system, types include Command Line (CLI), Graphics User Interface (GUI), and Batch (punch cards)

Program execution – The system must be able to load a program into memory and be able to run that program, end execution, either normally or abnormally (indicating errors)

I/O operations – A running program may require I/O, which may involve a file or an I/O device, the OS must provide support to these I/O mediums

File-system manipulation – The file system is of particular interest. Programs need to read and write files and directories, create and delete them, search them, list file information, and manage permissions and user access levels.

Communications – Processes may exchange information, on the same computer or between computers over a network. Communications may be via shared memory or message passing.

Error Detection – OS needs to be constantly aware of possible errors. They may occur in the CPU/memory hardware, I/O devices, or in a user program. For each type of error the OS should take appropriate action and (if possible) allow debugging facilities.

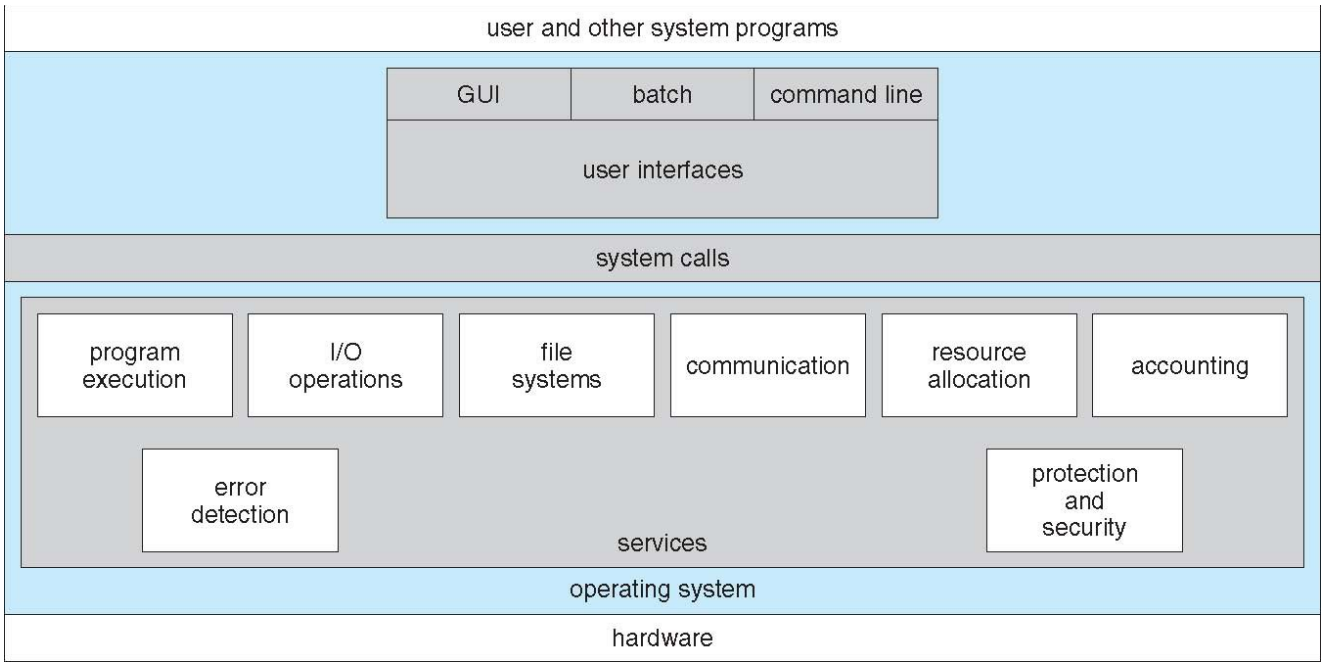
Resource allocation – as discussed in Chapter 1, the OS is a resource allocator, it needs to allocate resources to multiple running jobs or users or many types (including CPU cycles, main memory, file storage and I/O devices).

Accounting – OS needs to keep track of which users use how much and what computer resources

Protection and Security – Owners of info stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other

protection involves ensuring that all access to system resources is controlled

security of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts



Know system calls: What are system calls, How is an system call implemented (Figure on the slide 2.17), Parameter passing, Three common API to system call.

System Calls are the programming interface to the services provided by the OS. Theses are typically written in high-level language (like C or C++) and accessed via a Application Programming Interface (API) rather than direct system call use.

How is it implemented : Typically, a number is associated with each system call. The System-Call interface maintains a table indexing according to these numbers, invoking intended system call in OS kernel and returning status of the system call and any return values.

Parameter passing is needed when more information than the identity of the call is needed (more often than not the case). Three general methods are used to pass parameters to the OS:

- simplest – pass parameters in registers
- parameters stored in a block, or table, in memory, and address of block passed as a parameter in a register
- parameters placed, or pushed, onto the stack by the program and popped off the stack by the OS

The three most common API to system call are Win32 API (windows), POSIX API (posix systems like linux and mac osx) and Java API (java VMs)

Know the difference between "policy" and "mechanism".

Policies are ways to choose which activities to perform. Mechanisms are the implementations that enforce policies, and often depend to some extent on the hardware on which the operating system runs. Simply, a policy is WHAT will be done and a mechanism is HOW to do it.

Know the various operating system structures: Monolithic, Layered, Microkernel, Loadable.

Monolithic Kernel – is an operating system architecture where the entire operating system is working in kernel space and is alone in supervisor mode. Essentially, it's not divided into modules but it's interfaces and levels of functionality are not well separated. (ex MS-DOS)

Layered Approach – The operating system is divided into a number or layers (levels), each built on top of lower layers. The bottom layer (zero) is the hardware; the highest (layer N) is the user interface. Each layer is selected such that each uses functions (operations) and services of only lower-level layers.

Microkernel – moves as much from the kernel into user space. Communication takes place between user modules via message passing. This is easier to extend and port the OS as well as more reliable/secure however it has a performance overhead from all the communication.

Loadable Kernel Modules – Uses a object oriented approach allowing each core component to be separate and talk over known interfaces; loads as needed within kernel. Overall it is similar to layers but more flexible.

Know some common operating systems' structures: Linux, Windows, Mac OS X.

Mac OS X: hybrid- layered with below layers as a microkernel environment.

Windows : is monolithic kernel structure.

Linux: has a loadable kernel structure.

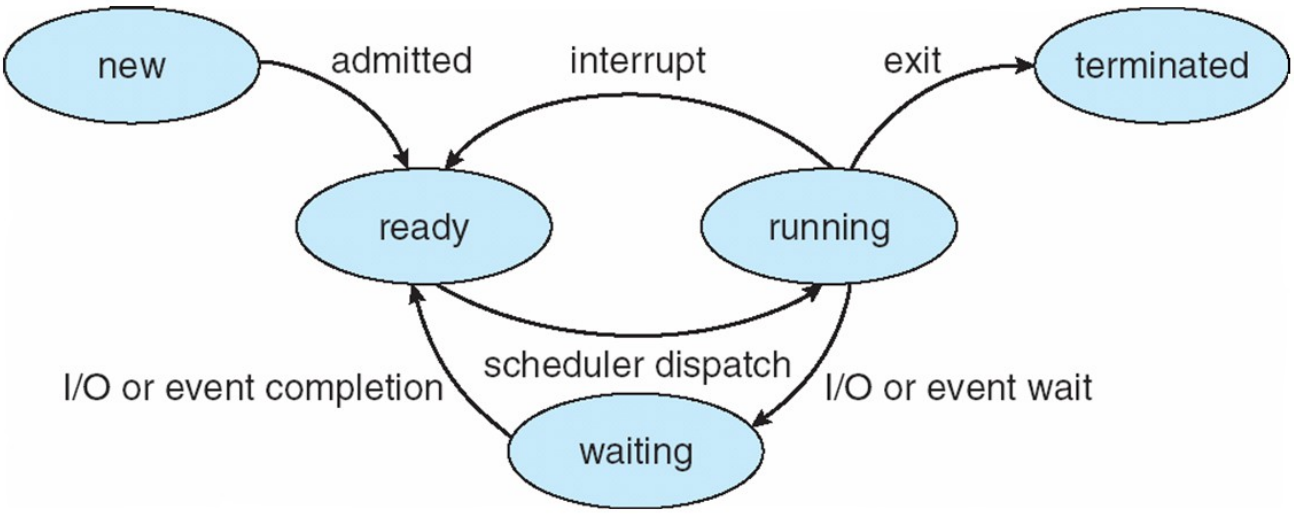
Chapter 3

Know process concepts: What is a process, Process components in memory,

Process – a program in execution; process execution must progress in sequential fashion

Components in Memory : Includes the program code; called the text section
Current activity including program counter, aka processor registers
Stack containing temp data
Data section containing global variables
Heap containing dynamic memory allocated during run time
Keep in mind that a program is the passive state of entity stored on the disk while the process is the active version.

Understand the transmission of process states. Figure on the slide 3.8.



Know PCB and at least 5 components of PCB.

PCB is the Process Control Block (also called the task control block) that is a data structure containing the information needed to manage the scheduling of a particular process.
Components include:

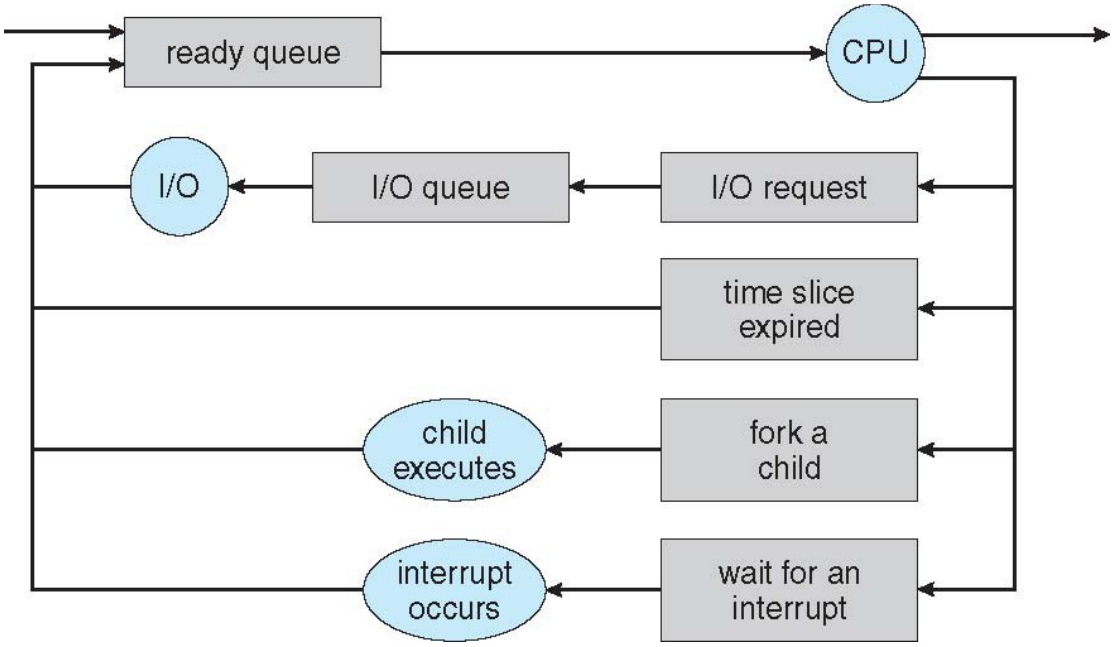
- Process state – running, waiting, etc
- Program counter – location of instruction to next execute
- CPU registers – contents of all process registers
- CPU scheduling info – priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits

I/O status information – I/O devices allocated to process, list open files

Understand the queues in process scheduling, Figure on the slide 3.15.

Process Scheduling allows us to maximize CPU use, quickly switching processes onto CPU for time sharing. The Process scheduler itself selects among available processes for next executing on the CPU

The queues for scheduling processes include:
Job queue – set of all processes in the system
Ready queue – set of all processes residing in main memory, ready and waiting to execute
Device queues – set of processes waiting for an I/O device



Know the differences of various process schedulers: short-term, long-term, medium-term.

Short-term scheduler (CPU scheduler) – selects which process should be executed next and allocates CPU. Sometimes this is the only scheduler on the system and is invoked frequently (milliseconds)

Long-term scheduler (Job scheduler) – selects which processes should be brought into the ready queue. It is invoked infrequently (mins/seconds) and controls the degree of multiprogramming

Medium-term scheduler – can be added if degree of multiple programming needs to decrease. This removes process from memory, stores on disk, and brings it back from disk to continue execution (swapping)

Understand context switch.

Context Switch – 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 via a context switch . Here, context of a process is represented in the PCB. This is a time overhead, the system does no useful work while switching.

Know the basic APIs: fork(), exec(), exit(), abort(), wait(), getPid(), and their corresponding

windows APIs.

fork() - creates a child process; windows equivalent is CreateProcess()
exec() - replaces the current process image with a new process image; windows equivalent is Call()
exit() - terminates the calling process immediately ;windows equivalent is ExitProcess()
abort() - abort current process, producing abnormal program termination; windows equivalent is TerminateThread()
wait() - wait for child process to exit; windows equivalent is WaitForSingleObject()
getPid() - get the process id of current running process; windows equivalent is GetCurrentProcessID()

Understand interprocess communication: shared memory, message passing

Interprocess communication is simply the communication of two process, methods include:

Shared Memory: An area of memory shared among the processes that wish to communicate. The communication is under the control of the users

Message Passing: Mechanism for processes to communicate and to synchronize their actions; processes communicate with each other without resorting to shared variables. IPC facilities provides two operations: send(message) and receive(message). The message size is either fixed or variable.

Communications in Client-Server Systems.

Processes must name each other explicitly:

send(P, message) – send a message to process P.

receive (Q, message) – receive a message from process Q

Properties of communication link:

links are established automatically. A link is associated with exactly one pair of communicating processes. Between each pair there exists exactly one link. The link may be unidirectional but is usually bi-directional

Chapter 4

Understand thread concepts: What is a thread, Why is it light weight.

Thread – A lightweight version of a process. Specifically, a small sequence of programmed instruction that can be managed independently by a scheduler.

Know the differences between parallelism and concurrency.

Parallelism - implies a system can perform more than one task simultaneously

Concurrency – supports more than one task making progress ie treats single process / core, scheduler providing concurrency

Know the three multithreading models: Many-to-One, One-to-One, Many-to-Many (two level). Know at least one example system in each model.

Many-to-One – many user-level threads mapped to single kernel thread. Few systems use this model as one thread blocking causes all to block. Example = Solaris Green Threads

One-to-One – Each user-level thread maps to kernel thread (number of threads restricted due to overhead). Creating a user-level thread creates a kernel thread. Example = Windows/Linux

Many-to-Many – Allows user-level threads to be mapped to many kernel threads. Allows the OS to create a sufficient number of kernel threads. Example = Windows with the ThreadFiber package

Two-level Model – similar to Many to Many, except allows a user thread to be **bound** to kernel thread. Example = HP - UX

Know the three primary thread libraries and the basic operations: pthread_create(), pthread_join(),pthread_exit(), and their corresponding operations in windows library and java library if there are.

Thread libraries provide a programmer with API for creating and managing threads. The three common thread libraries are Pthreads, Windows Multithreading and Java Threads (managed by the JVM).

pthread_create() - starts a new thread in the calling process; Windows = CreateProcess()

pthread_join() - suspends execution of calling thread until target thread terminates; Windows = WaitForSingleObject()

pthread_exit() - terminates calling thread; Windows = TerminateThread()

Understand threading issues: semantics of fork(), signal handling, thread cancellation, thread local storage, scheduler activations.

Semantics of fork() - Does fork() duplicate only the calling thread or all threads?
-Some unix systems have two versions of fork()
-exec() usually works as normal, replaces the running process including all threads

signal handling – Signals are using in UNIX to notify a process that a particular event has occurred.
- A signal handler is used to process signals
  - Signal is generated by particular event
  - Signal is delivered to a processors
  - Signal is handed by one of two signal handlers: default or user-defined

thread cancellation – Terminating a thread before it has finished, with asynchronous cancellation it terminates the target thread immediately, while a deferred cancellation allows the target thread to periodically check if should be canceled. This may request cancellation but it actual depends on state of

the thread for when that might happen or how.

Thread local storage – (TLS) allows for each thread to have its own copy of data. Useful when you do not have control over the thread creation process. Similar to static data, as TLS is unique to each thread.

Scheduler Activations – both Many to Many and Two-level models require communication to maintain the appropriate number of kernel threads allocated to the application. Typically, they use a intermediate data structure between user and kernel threads (Lightweight Process (LWP). Scheduler activations provide upcalls – a communication mechanism from the kernel to the upcall handler in the thread library. This communication allows an application to maintain the correct number of kernel threads.

Windows threads and Linux threads

Windows threads – implement one-to-one mapping, kernel-level
Each thread contains

- A thread id
- Register set representing state of thread
- Separate user and kernel stacks for when thread runs in user mode or kernel mode
- Private data storage area used by run-time libraries and dynamic link libraries (DLLs)

The register set, stacks, and private storage area are known as the context of the thread

Linux threads – refers to threads as tasks and thread creation is done with the clone() system call. Additionally, clone() allows a child tasks to share the address space of the parent task.

Chapter 5

Know basic concepts: Process execution cycle, short-term scheduler, preemptive scheduling, non-preemptive scheduling, dispatch latency,

Process execution cycle includes a cycle of CPU execution and I/O wait, specifically a CPU burst followed by I/O burst. CPU burst distribution is of main concern.

Short-term scheduler – selects from among the processes in ready queue, and allocates the CPU to one of them. Switching processes from waiting, ready, and running as well as termination. These types of scheduling is non-preemptive, while considering access to shared data, preemption while in kernel mode, and interrupts occurring during crucial OS activities is preemptive.

Dispatch latency – time it takes for the dispatcher to stop one process and start another running.

In non-preemptive once a process start its execution it will terminate only when it ends while in preemptive a process can be interrupted by another process in mid of its execution also.

Know scheduling criteria: CPU utilization, Throughput, Turnaround time, Waiting time, Response time and Know how to calculate a schedule under those criteria

CPU utilization – keep the CPU as busy as possible
Throughput – # of processes that complete their execution per time unit
Turnaround time – amount of time to execute a particular process (end time – start time)

Waiting time – amount of time a process has been waiting in the ready queue (ie, time a process sits after being put in the queue before first execution)
Response time – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

Understand scheduling algorithms: FCFS, SJF, Priority, RR, Multilevel (feedback) Queue

First – Come, First – Served (FCFS) – The first process put in the ready queue is the the first process to finish execution, ie, processes executing in order of arrival. This gives minimum average wait time for a set of given processes

Shortest Job First (SJF) – The first process in the ready queue is the process with the shortest CPU burst time. This is considered a non-preemptive scheduler since it is hard to know length of a job before hand, we can usually only estimate. The preemptive version is called shortest-remaining-time-first and changes order in queue for whatever job has the smallest CPU burst time at that given point of execution

Priority Scheduling – A priority number (integer) is associated with each process. The CPU is allocated to the process with the highest priority. Keep in mind that SJF is priority scheduling where priority is the inverse of predicted next CPU burst time. This has the problem of starvation that can be solved with Aging.

Round Robin (RR) – Each process gets a small unit of CPU time (time quantum q). After this time has elapsed the process is preempted and added to the end of the ready queue. A timer iterrupts every quantum to schedule next process. Typically this has a higher turnaround than SJF, but better response time.

Multilevel Queue – ready queue is partitioned into separate queues ie, foreground (interactive) and background (batch). Process is permanently in a given queue and each queue has it's own scheduling algorithm. Scheduling must be done between queues.

Multilevel Feedback Queue – In the feedback version of a multilevel queue, a process can move between the various queues, aging can be implemented this way. Queues can have priority to execute over others as well. It's kind of like “If not finished in queue 1 with time quantum, move to queue 2” and so forth based on various conditions.

Know how to draw a Gantt Chart of a schedule under above scheduling algorithms

Understand starvation and its solution

Starvation is the result of queues with priorities over other processes, ie, low priority processes may never execute in a queue with constant higher priority elements coming in. This is solved by Aging, or increasing the priority of a process as time progress/it stays out of execution.

Know concepts: asymmetric multiprocessing, symmetric multiprocessing (SMP)

Asymmetric multiprocessing – only one processor accesses the system data structures, alleviated the need for data sharing.

Symmetric multiprocessing (SMP) – each processor is self-scheduling, all processes in common ready queue, or each has its own private queue of ready processes (current most common scheduling)

Understand processor affinity, load balancing, push migration, pull migration

Processor affinity – process has affinity for processor on which it is currently running (ie runs best on it)

Load balancing – attempts to keep a workload evenly distributed between multiple processors

Push migration – periodic task checks load on each processor and if found pushes task from overloaded CPU to other CPUs

Pull migration – idle processors pull waiting task from busy processor.

Chapter 6

Know basic concepts: race condition, atomic operation, busy waiting, spinlock

Race Condition – occurs when two or more threads can access shared data and they try to change it at the same time. Because the tread scheduling algorithm can swap between threads at any time, you don't know the order in which the threads will attempt to access the shared data.

Atomic operation – Non-interruptible operations, ie, it appears to the rest of the system to occur at once, without being interrupted. This is the guarantee of isolation from interrupts, signals, and concurrent processes and threads.

Busy waiting – When a process repeatedly checks to see if a condition is true, such as if a lock is available, or a certain keyboard input is given.

Spinlock – a lock that causes a thread trying to acquire it to simply wait in a a loop (“spin”) while it repeatedly checking if the lock is available, since the thread remains active but is not performing a useful task, the use of such a lock is a kind of busy waiting.

Understand process synchronization and why it is necessary

The idea of Process Synchronization means sharing system resources by processes in such a way that, concurrent access to shared data is handled thereby minimizing the chance of inconsistent data. It is necessary to handle issues that arose when multiple process executions occur. Problems like the Critical-section problem.

Know what is critical-section and critical-section problem as well as three conditions to be filled to solve problem

The critical-section problem is that in a system of given processes, each process has a critical section of code that can only be accessed by one process at a time, ie, when one process is in a critical section, no

other may be in its critical section. There in lies the problem, how to get around or format for this issue to prevent lost CPU time among other things. Each process must ask permission to enter critical section in entry section, may follow critical section with exit section, then remainder section.

Conditions a solution must fulfill :

1. Mutual Exclusion - If a process is executing its critical section, then no other processes can be executing their critical sections
2. Progress – If no process is executing in its critical section, and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely
3. Bounded Waiting – A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to ether its critical section and before that request is granted

Know what is Peterson's solution and hardware solution

The Peterson's solution for the critical-section problem is a good algorithmic description of solving the problem and a two process solution. Assume that the load and store machine-language instructions are atomic; that is, cannot be interrupted. The two process share the variables of an int called turn and a Boolean called flag[2]. The variable turn indicates whose turn it is to enter the critical section. While the variable flag array is used to indicate if a process is read to enter it's own critical section. These two elements allow us to solve the problem while fulfilling all conditions of a solution.

The hardware solution is Synchronization Hardware. This is based on the idea of locking, or protecting critical regions via locks. This locks are and their operations are atomic or non-interruptible. Essentially, this allows locks to prevent and manage access to critical sections until they should be accessing those sections in a determined sequence.

Understand Mutexes, Semaphore, and Monitor, know the differences between them

Mutex - is a lock that we set before using a shared resource and release after using it When the lock is set, not other thread can access the locked region of code. This ensures a synchronized access of the shared resources in the code. It is binary, thus is just one single lock.

Semaphore – is a synchronization tool that provides more sophisticated ways (than Mutex locks) for processes to synchronize their activities. Main difference being that it can be both binary (just like a mutex lock) or be a Counting Semaphores, meaning it can hold a integer value over an unrestricted domain, essentially multiple 'locks' or 'reasources' can be held in one semaphore.

Monitor – an abstract data type which only allows one process to execute in the critical section at a time. It has internal variables that can only be accessed by by within the procedure.

Know deadlock, starvation, and priority inversion problem in semaphore usage

Deadlock – two or more processes are waiting indefinitely for an event that can be cause by only one fo the waiting processes

Starvation – A process may never be removed from the semaphore queue in which it is suspended

Priority Inversion – Scheduling problem when lower-priority process holds a lock needed by high-priority process. This would mean in the case of a semaphores one or more processes is using the resources needed by a higher priority process. We can using aging and priority inheritance to solve these issues.

Understand the three classical synchronization problems: Bounded-Buffer, Readers and Writers, and Dining Philosopher (be able to fill in blanks if partial solution is given)

Bounded-Buffer – In which you have n buffers that each can hold one item, made by a producer and removed by a consumer. The simple solution is to have three locks, one for signaling which process is in use with value 1(the consumer or the producer). One for setting the buffer to full(value 0) and finally one for setting it to empty(value n). That way you can produce until full/done and consume until needing to stop/the buffer is empty without both processes going at the same time.

Readers and Writers – A data set is shared among concurrent reader and writer processes. Readers only read while Writers read and make edits. Only one writer can access the data set at a time. A simple implementation is to have the data set, a read and write lock (set 0), a simple lock to indicate if readers are maxed, and finally a count for the number of readers. This allows the rw to be locked if a reader or readers are in use of the process, but can allow the writer/writers to quickly access the data ahead of the reader for edits.

Dining-Philosopher – In this problem we have a table of philosophers that can either be thinking or eating, never interacting with neighbors but sharing chopsticks. They pick up chopsticks one at a time to eat (one from each side) and thus it must be made sure that we never have a deadlock for chopsticks. Thus we need to have a semaphore with n chopsticks (in the case of n philosophers, usually an odd number). We can handle the deadlocking by:

1) Allowing at most n – 1 philosophers to be sitting simultaneously to eat
2) Allowing a philosopher to pick up only if both chopsticks are available
3) Using an asymmetric solution, an odd-numbered philosopher picks up the chopsticks left first while an even-numbered philosopher picks them up right first

Chapter 7

Know basic concepts: system resources, request edge, assignment edge, claim edge, safe state, unsafe state

System resources – a series of finite resources in a computer system that can be distributed among some number of competing processes. Can be one of many types or one of a group of types.

Request edge – the indication that a processes has requested a resource. In a resource allocation graph this is a dotted line line. A transformation of a claim edge.

Assignment edge – the indication that a resource is currently allocated to a process. In a resource allocation graph this is a bold line. A transformation of a request edge

Claim edge – the resources a process could possibly make a request to. In a resource-allocation graph it

is shown by being in the same space of a resource in the graph (ie the process is part a of graph with resource x).

Safe state – a state in which all processes in the system can still request resources that can be satisfied by the currently available resources.

Unsafe state – a state in which one more more processes in the system could request a reasource that is not currently available from the remaining resources free.

Understand the four conditions for deadlock

Mutual exclusion: only one process at a time can use a resource

Hold and wait: a process holding at least one resource is waiting to acquire additional resources held by other processes

No preemption: a resource can be released only voluntarily by the process holding it, after that process has completed its task

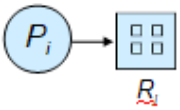
Circular wait: there exists a set $\{P_0, P_1, \ldots, P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by P_2, …, P_{n-1} is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0.

Know resource-allocation graph and how to draw a resource-allocation graph

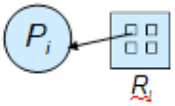
- Process

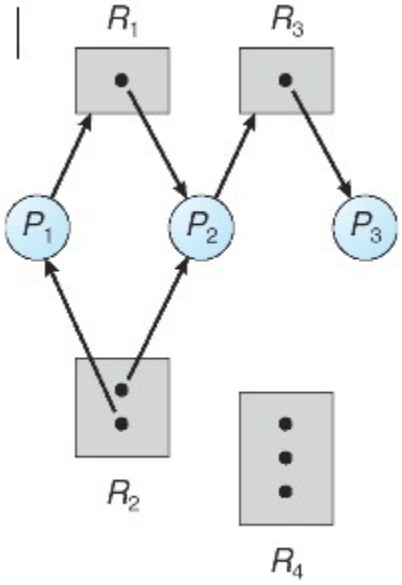
- Resource Type with 4 instances

- P_i requests instance of R_j



- P_i is holding an instance of R_j





Know how to check if there is a deadlock in a resource-allocation graph (single instance resource, multiple instances resource)

If graph contains no cycles ⇒ no deadlock

If graph contains a cycle ⇒
    if only one instance per resource type, then deadlock
    if several instances per resource type, possibility of deadlock

Know methods to handle deadlocks: prevention, avoidance, detection and recovery, or ignore

Deadlock prevention – restrains the ways requests can be made, so that the possibility of a deadlock is not possible. This is done by guaranteeing processes don't hold other resources when trying to execute and imposing an order to requesting resources by type. Basically making processes stop execution until all for their resources would be made available.

Deadlock avoidance – Checks the maximum number of resources each process to be executed needs (ie needs preemptive information) and orders execution to avoid deadlocks when they could occur.

Deadlock Detection – Algorithm used in an avoidance system to check if a deadlock could occur, generally run in such a way that prevents as much processor rollback as possible.

Deadlock Recovery – Done via process termination, abort all deadlock processes and then one running process at a time until the deadlock cycle is eliminated.

Deadlock Ignore – Assume a deadlock will not occur in an unsafe state and continue executions without running into the case of a deadlock.

Understand deadlock prevention: violate one deadlock condition, ordering lock

Mutual exclusion is prevented because non-shareables must be held

Hold and wait is prevented because holding is only allowed when a process already holds a resource, ie all resources must be allocated before execution or allow processes to request resources only when it has none.

No preemption is prevented by forcing a process to drop all resources upon requesting a resource unavailable where preempted resources are added to the list of resources for which the process is waiting and the process will only restart when all resources (old and new) are available.

Circular wait is prevented with the ordering lock, or ordering of all resource types, and each process must follow this order of requests.

Understand deadlock avoidance, slide 7.21

Try to keep the program in the safe state where no deadlocks can occur, if at any point we reach the unsafe state fix the issue! IE Avoidance = ensure that a system can never enter the unsafe state where there is the possibility of a deadlock.

Know avoidance algorithms: resource-allocation graph and banker's algorithm, Can use these algorithms given a system with processes

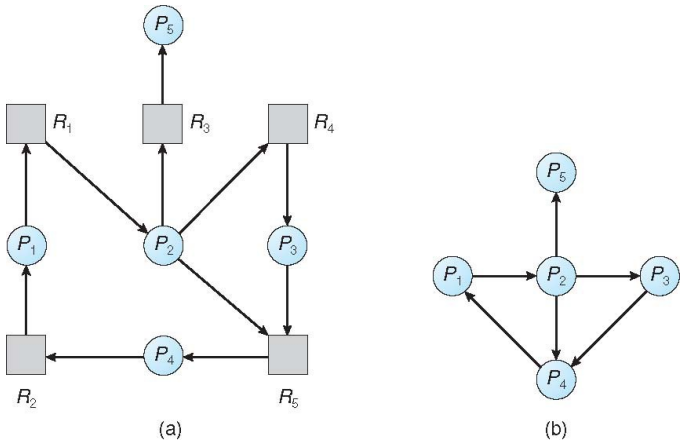
In the case of a single instance of a resource a resource-allocation graph, with multiple resource types use the banker's algorithm.

KEY TO KNOW(RESOURCE-ALLOCATION)

The request can be granted only if converting the request edge to an assignment edge does not result in the formation of a cycle in the resource allocation graph

KEYS TO KNOW(BANKERS):
Need = Max – Allocation
Max = Allocation + Need
Available = Resources Available - Total Allocation
Work Starts at Available and has Allocation added from each ending process
If Work >= Max, the process can end execution and it's current allocation is added to work
Match each point and mark as true (adding to work) until all processes run or work cannot become bigger than the Max need of a process, meaning there is a deadlock.

Understand deadlock detection: wait-for graph



On Left Resource-allocation Graph

On Right Corresponding wait-for Graph

ALL THIS IS PROCESS EXEC ORDER

Know detection algorithm and know how to use it

It's Bankers

Understand deadlock recovery: select a victim and rollback

Selects the victim that would have the minimum cost and rollback, or return to some safe state, restarting processes for rollback. Since some process might always be picked a the victim starvation is a possible issue.

Chapter 8

Know basic concepts: logical memory, physical memory, MMU, page, frame, and so on

Logical Memory – is the address space, assigned to a logical partition, that the operating system perceives as its main storage. These logical address spaces are defined by base and limit registers.

Physical Memory – The physical memory taken up on the actual device that the logical memory is

mapped to.

MMU – Memory-Management Unit, a hardware device that, at run time, maps virtual to physical address works by the use of adding the relocation register to the base register.

Page – A fixed length contiguous block of virtual memory, describing a single entry in a page table.

Frame – A fixed block of consecutive physical memory. Associated with pages.

Understand contiguous memory allocation, noncontiguous memory allocation, external fragmentation, internal fragmentation

Contiguous Allocation – A memory allocation model that assigns a process to consecutive memory blocks (that is, memory blocks having consecutive addresses). When a process needs to be executed, memory is requested by the process. The size of the process compared with the amount of contiguous main memory available to execute the process. If sufficient contiguous memory is found, the process is allocated memory to starts its execution. Otherwise, it is added to a queue of waiting processes until sufficient free contiguous memory is available.

Noncontiguous Allocation – A memory allocation model that assigns the separate memory blocks at the different location in memory space in a nonconsecutive manner to a process requesting for memory.

External Fragmentation – Holes in the memory allocation or disk space caused by processes freeing from memory or just unallocated space. In general, these are free memory space, but might be too small for any use. In other words its total memory space that exists to satisfy a request, but it is not contiguous

Internal Fragmentation – Holes in memory caused by rounding up from actual requested allocation to allocation granularity. In general, these are caused by processes needing extra space but not using it. In other words it is the size difference that is memory internal to a partition not being used.

Know how to convert from the logical address to physical address for segmentation, paging, hierarchical paging

For Segmentation: The segment table maps two-dimensional physical addresses by their base (physical address) and limit (length). This table has it's own base register which points to it's address in memory and it's own length register that indicates the number of segments used by a program. Each entry in the table also has a validation bit and read/write/execution privileges. This memory allocation is of course, dynamic. To be put into the table and given a physical address, it must compare the limit to the table length to see if it can fit, then it is mapped to the table and thus physical memory.

For Paging: Paging can be noncontinuous (ie without touching borders or split at different places) but avoids external fragmentation and varying sized memory chunks when possible. It maps pages to frames that are free (unused). Ie, to run a program of size N pages it needs N free frames to load the program. It translates logical to physical memory address with a page table. The page number is an index in the table containing the base address and the page offset defines the physical memory address sent to the memory unit. Essentially logical addresses are given to the table and the table gives them physical addresses. Basically we can look for where the memory is physically that we need by looking at the corresponding page table entry.

For Hierarchical: Breaks up the logical address space into multiple page tables, a simple technique is a two-level page table where we page the page table. Thus we split the logical address into to locations with the same offset.

Know how to convert a address space to page representation

Chapter 17

Know basic concepts: distributed system, network system, LAN, WAN, and so on

Distributed system – is a collection of loosely coupled processors interconnected by a communications network

Network System – or Network operating system is a system in where users are aware of multiplicity of machines and access to various machines is done explicitly.

LAN – Local Area Network, designed to cover small geographical area

WAN – Wide Area Network, links geographically separated sites

Know how Ethernet works, how token ring works

Ethernet is a multi-access bus-based connection that can be used to construct a LAN network. To be specific it is a system for connecting a number of computer systems to form a LAN, with protocols to control the passing of information and to avoid simultaneous transmission by two or more systems. It works by connecting multiple computers or peripherals by cable data-bus (Ethernet cable) to a LAN switch, which can in turn be passes into a firewall to a router to give the LAN network WAN access. Essentially, in communication, each host needs to listen to the bus, if empty it can communicate, otherwise it must wait.

A token ring is a LAN technology that is a communications protocol for LAN. It uses a special three-byte frame balled a “token” that travels around a logical “ring” of workstations or servers. This token passing is a channel access method providing fair access for all stations and eliminating the collisions of contention-based access methods.

Understand problems and solutions in communication: addressing problem, routing problem, connection problem

Addressing problem – How do two processes locate each other to communicate?

> -This can be solved in various ways, such as naming systems in the network, addressing messages with process id, identifying processes on remote systems via <host-name, identifier> pair, and DNS or Domain Name System which specifies host names and address resolution.

Routing problem – How are messages sent through the network?

> -Can be sent either with Fixed routing (Paths made in advance), Virtual routing (paths made per messaging session) or Dynamic routing (message path made only when a message is sent). Each has it's own trade-offs with speed of connecting, passing speed, and transfer speed.

Connection problem – How do two processes send a sequence of messages?

-Can use circuit switching (permanent physical link is established for duration of communication) message switching (A temporary link is established for the duration of one message transfer) or packet switching (messages are divided into fixed length packages and sent to destination) to solve this issue.

Know OSI 7-layers and TCP/IP protocol stack

The layers of OSI (Communication protocol)

Layer 1: Physical layer – handles the mechanical and electrical details of the physical transmission of a bit stream

Layer 2: Data-link layer – handles the *frames* , or fixed- length parts of packets, including any error detection and recovery that occurred in the physical layer

Layer 3: Network layer – provides connections and routes packets in the communication network, including handling the address of outgoing packets, decoding the address of incoming packets, and maintaining routing information for proper response to changing load levels

Layer 4: Transport layer – responsible for low-level network access and for message transfer between clients, including partitioning messages into packets, maintaining packet order, controlling flow, and generating physical addresses

Layer 5: Session layer – implements sessions, or process-to- process communications protocols

Layer 6: Presentation layer – resolves the differences in formats among the various sites in the network, including character conversions, and half duplex/full duplex (echoing)

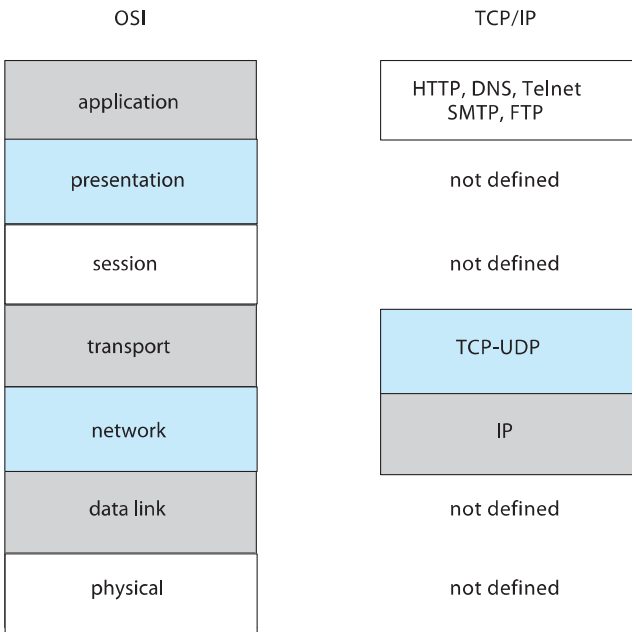
Layer 7: Application layer – interacts directly with the users, deals with file transfer, remote-login protocols and electronic mail, as well as schemas for distributed databases

The layers of the TCP/IP protocol stack

Layer 1: Network Interface Layer – (IP)

Layer 2: Host-to-Host Transport Layer - (TCP - UDP)

Layer 3: Application Layer – (HTTP, DNS, Telnet, SMTP, or FTP)


OSI
TCP/IP
application
presentation
session
transport
network
data link
physical
HTTP, DNS, Telnet
SMTP, FTP
not defined
not defined
TCP-UDP
IP
not defined
not defined