Chapter 1

- Computer system can be divided into four components
  - Hardware – provides basic computing resources
    - CPU, memory, I/O devices
  - Operating system
    - Controls and coordinates use of hardware among various applications and users
  - Application programs – define the ways in which the system resources are used to solve the computing problems of the users
    - Word processors, compilers, web browsers, database systems, video games
  - Users
    - People, machines, other computers

- Kernel: The one program running at all times on the computer

- bootstrap program is loaded at power-up or reboot
  - Typically stored in ROM or EPROM, generally known as firmware
  - Initializes all aspects of system
  - Loads operating system kernel and starts execution

- Interrupt driven by hardware
- Software error or request creates exception or trap
- interrupt vector, which contains the addresses of all the service routines
- type of interrupt:
  - polling
  - vectored
- System call: request to the OS to allow user to wait for I/O completion
- Device-status table: contains entry for each I/O device indicating its type, address, and state

Storage Structure

- Main memory
  - Random access
  - Typically volatile
- Secondary storage
  - large nonvolatile storage capacity
- The most common secondary storage device is Magnetic disks
- Magnetic disks
  - Disk surface is logically divided into tracks, which are subdivided into sectors
  - The disk controller determines the logical interaction between the device and the computer
- Solid-state disks: faster than magnetic disks, nonvolatile

- ❖ Caching: copying information into faster storage system; main memory can be viewed as a cache for secondary storage
- ❖ Device Driver:
  - for each device controller to manage I/O
  - Provides uniform interface between controller and kernel

- ❖ Multiprocessors systems growing in use and importance
  - Also known as parallel systems, tightly-coupled systems
  - Advantages include:
    1. Increased throughput
    2. Economy of scale
    3. Increased reliability – graceful degradation or fault tolerance
  - Two types:
    1. Asymmetric Multiprocessing
    2. Symmetric Multiprocessing

Clustered Systems

- ❖ Like multiprocessor systems, but multiple systems working together
- ❖ Usually sharing storage via a storage-area network (SAN)
- ❖ Provides a high-availability service which survives failures
  - Asymmetric clustering has one machine in hot-standby mode
  - Symmetric clustering has multiple nodes running applications, monitoring each other
- ❖ Some clusters are for high-performance computing (HPC)
  - Applications must be written to use parallelization
- ❖ Some have distributed lock manager (DLM) to avoid conflicting operations

Operating System Structure

- ❖ Multiprogramming needed for efficiency
  - Single user cannot keep CPU and I/O devices busy at all times
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
  - A subset of total jobs in system is kept in memory
  - One job selected and run via job scheduling
  - When it must wait (for I/O for example), OS switches to another job
- ❖ Timesharing (multitasking) is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating interactive computing
  - Response time should be < 1 second
  - Each user has at least one program executing in memory ► process
  - If several jobs ready to run at the same time ► CPU scheduling
  - If processes don't fit in memory, swapping moves them in and out to run

  - Virtual memory allows execution of processes not completely in memory

Operating-System Operations

- Dual-mode operation allows OS to protect itself and other system components
  - User mode and kernel mode
  - Mode bit provided by hardware
    - Provides ability to distinguish when system is running user code or kernel code
    - Some instructions designated as privileged, only executable in kernel mode
    - System call changes mode to kernel, return from call resets it to user

Process Management

- A process is a program in execution. It is a unit of work within the system. Program is a passive entity, process is an active entity.
- Single-threaded process has one program counter specifying location of next instruction to execute

Process Management Activities

- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication
- Providing mechanisms for deadlock handling

Memory Management:

- All data in memory before and after processing
- All instructions in memory in order to execute
- Memory management determines what is in memory when
  - Optimizing CPU utilization and computer response to users
- Memory management activities
  - Keeping track of which parts of memory are currently being used
  - Deciding which processes (or parts thereof) and data to move into and out of memory
  - Allocating and deallocating memory space as needed

Storage Management:

- OS provides uniform, logical view of information storage
  - Abstracts physical properties to logical storage unit - file
  - Each medium is controlled by device (i.e., disk drive, tape drive)
    - Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)
- File-System management
  - Files usually organized into directories
  - Access control on most systems to determine who can access what
  - OS activities include
    - Creating and deleting files and directories
    - Primitives to manipulate files and dirs

        - Mapping files onto secondary storage
        - Backup files onto stable (non-volatile) storage media

Protection and Security:

- **Protection** – any mechanism for controlling access of processes or users to resources defined by the OS
- **Security** – defense of the system against internal and external attacks
    - Example: Huge range, including denial-of-service, worms, viruses.
- Systems generally first distinguish among users, to determine who can do what
    - User identities (**user IDs**, security IDs) include name and associated number, one per user
    - User ID then associated with all files, processes of that user to determine access control
    - Group identifier (**group ID**) allows set of users to be defined and controls managed, then also associated with each process, file
    - **Privilege escalation** allows user to change to effective ID with more rights

Computing Environments

- **Traditional**
    - Stand-alone general purpose machines
    - **Portals** provide web access to internal systems
    - **Network computers** (**thin clients**) are like Web terminals
    - Mobile computers interconnect via **wireless networks**
    - Networking becoming ubiquitous – even home systems use **firewalls** to protect home computers from Internet attacks
- **Mobile**
    - Handheld smartphones, tablets.
    - Extra feature – more OS features (GPS, gyroscope)
    - Allows new types of apps like augmented reality
    - Use IEEE 802.11 wireless, or cellular data networks for connectivity
    - Leaders are Apple iOS and Google Android
- **Distributed**
    - Collection of separate, possibly heterogeneous, systems networked together
    - **Network** is a communications path, **TCP/IP** most common
    - **Local Area Network (LAN)**
    - **Wide Area Network (WAN)**
    - **Metropolitan Area Network (MAN)**
    - **Personal Area Network (PAN)**
    - **Network Operating System** provides features between systems across network
- **Client-Server Computing**
    - Dumb terminals supplanted by smart PCs
    - Many systems now **servers**, responding to requests generated by **clients**
    - **Compute-server system** provides an interface to client to request services (i.e., **database**)

- **File-server system** provides interface for clients to store and retrieve files

- **Peer-to-Peer**
  - Another model of distributed system
  - P2P does not distinguish clients and servers
  - Instead all nodes are considered peers
  - May each act as client, server or both
  - Node must join P2P network
- **Virtualization**
  - Allows operating systems to run applications within other OSes
    - Vast and growing industry
  - **Emulation** used when source CPU type different from target type (i.e. PowerPC to Intel x86)
- **Cloud Computing**
- Delivers computing, storage, even apps as a service across a network
- Logical extension of virtualization as based on virtualization
- Types:
- **Public cloud** – available via Internet to anyone willing to pay
- **Private cloud** – run by a company for the company's own use
- **Hybrid cloud** – includes both public and private cloud components
- Software as a Service **(SaaS)** – one or more applications available via the Internet (i.e. word processor)
- Platform as a Service (**PaaS)** – software stack ready for application use via the Internet (i.e a database server)
- Infrastructure as a Service (**IaaS)** – servers or storage available over Internet (i.e. storage available for backup use)

Chapter 2

Operating System Services:

- ❖ Operating systems provide an environment for execution of programs and services to programs and users
- ❖ One set of operating-system services provides functions that are helpful to the user:
    - **User interface**: Almost all operating systems have a user interface(UI)
        - Varies between **Command-Line (CLI), Graphics User Interface (GUI), Batch**
    - **Program execution**: The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
    - **I/O operations**: A running program may require I/O, which may involve a file or an I/O device.
    - **File-system manipulation**: The file system is of interest.
    - **Communications:** Processes may exchange information, on the same computer or between computers over a network
    - **Error detection:** OS needs to be constantly aware of possible errors
        - **Debugging** facilities can greatly enhance the user's and programmer's abilities to efficiently use the system
    - **Resource allocation:** When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
    - **Accounting:** To keep track of which users use how much and what kinds of computer resources
    - **Protection and security:** The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other

Types of System Calls:

- ❖ Process control
- ❖ File management
- ❖ Device management
- ❖ Information maintenance
- ❖ Communications
- ❖ Protection

System Programs

- System programs provide a convenient environment for program development and execution. They can be divided into:
    - File manipulation
    - Status information sometimes stored in a File modification
    - Programming language support
    - Program loading and execution
    - Communications
    - Background services
    - Application programs
- **File management** - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
- **Status information**
    - Some ask the system for info - date, time, amount of available memory, disk space, number of users
    - Others provide detailed performance, logging, and debugging information
    - Some systems implement a **registry** - used to store and retrieve configuration information
- **File modification**
    - Text editors to create and modify files
    - Special commands to search contents of files or perform transformations of the text
- **Programming-language support**: Compilers, assemblers, debuggers and interpreters sometimes provided
- **Program loading and execution**: Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
- **Communications**: Provide the mechanism for creating virtual connections among processes, users, and computer systems
    - Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages.
- **Background Services:** Launch at boot time
- **Application programs:** Don't pertain to system, Run by users

System Boot:

- When power initialized on system, execution starts at a fixed memory location
    - **Firmware** ROM used to hold initial boot code
- Operating system must be made available to hardware so hardware can start it
    - Small piece of code – **bootstrap loader**, stored in **ROM** or **EEPROM** locates the kernel, loads it into memory, and starts it
    - Sometimes two-step process where **boot block** at fixed location loads bootstrap loader
- Common bootstrap loader, **GRUB**, allows selection of kernel from multiple disks, versions, kernel options
- Kernel loads and system is then **running**

Chapter 3

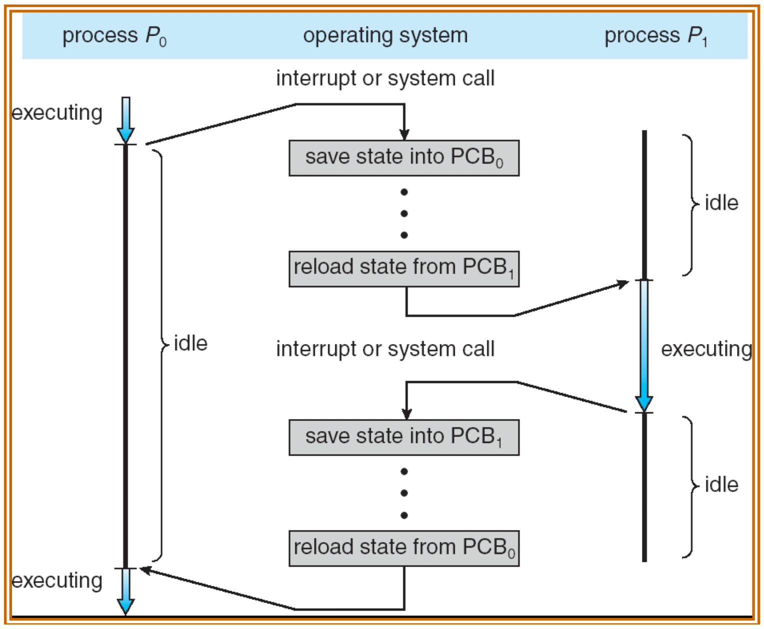
Process Concept

- An operating system executes a variety of programs:
  - Batch system – **jobs**
  - Time-shared systems – **user programs** or **tasks**
- Textbook uses the terms **job** and **process** almost interchangeably
- **Process** – a program in execution; process execution must progress in sequential fashion parts:
  - The program code, also called **text section**
  - Current activity including **program counter**, processor registers
  - **Stack** containing temporary data
  - **Data section** containing global variables
  - **Heap** containing memory dynamically allocated during run time
- Program is **passive** entity stored on disk (**executable file**), process is **active**
  - Program becomes process when executable file loaded into memory
- One program can be several processes

Process State:

- As a process executes, it changes *state*
  - **new**: The process is being created
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur
  - **ready**: The process is waiting to be assigned to a processor
  - **terminated**: The process has finished execution

CPU Switch from Process to Process:



Process Control Block (PCB) (task control block)

- **Process state**: running, waiting.
- **Program counter**: location of instruction to next execute
- **CPU registers**: contents of all process-centric registers
- **CPU scheduling information**: 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 of open files

Threads

- process has a single thread of execution
- Consider having multiple program counters per process
    - Multiple locations can execute at once
        - Multiple threads of control -> **threads**
- Must then have storage for thread details, multiple program counters in PCB

Process Scheduling

- Maximize CPU use, quickly switch processes onto CPU for time sharing
- **Process scheduler** selects available processes for next execution on CPU

scheduling queues of processes

- **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

Schedulers

- **Long-term scheduler** (or **job scheduler**) – selects which processes should be brought into the ready queue, invoked very infrequently (seconds, minutes), and controls the **degree of multiprogramming.**
- **Short-term scheduler** (or **CPU scheduler**) – selects which process should be executed next and allocates CPU, invoked very frequently (milliseconds).

Processes can be described as either:

- **I/O-bound process**: spends more time doing I/O than computations.
- **CPU-bound process**: spends more time doing computations.

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**

Google Chrome Browser is multiprocess with 3 categories

- **Browser** process manages user interface, disk and network I/O
- **Renderer** process renders web pages, deals with HTML, JavaScript, new one for each website opened

- **Plug-in** process for each type of plug-in

Synchronization

- **Blocking** is considered **synchronous**
  - **Blocking send** has the sender block until the message is received
  - **Blocking receive** has the receiver block until a message is available
- **Non-blocking** is considered **asynchronous**
  - **Non-blocking** send has the sender send the message and continue
  - **Non-blocking** receive has the receiver receive a valid message or null

Communications in Client-Server Systems:

- Sockets
- Remote Procedure Calls
- Pipes
- Remote Method Invocation (Java)

- **A socket** is identified by an IP address concatenated with a port number
- **A loopback** is a special IP address: 127.0.0.1. When a computer refers to IP address 127.0.0.1, it is referring to itself. When using sockets for client/server communication, this mechanism allows a client and server on the same host to communicate using the TCP/IP protocol.

Chapter 4

benefits of multithreaded programming

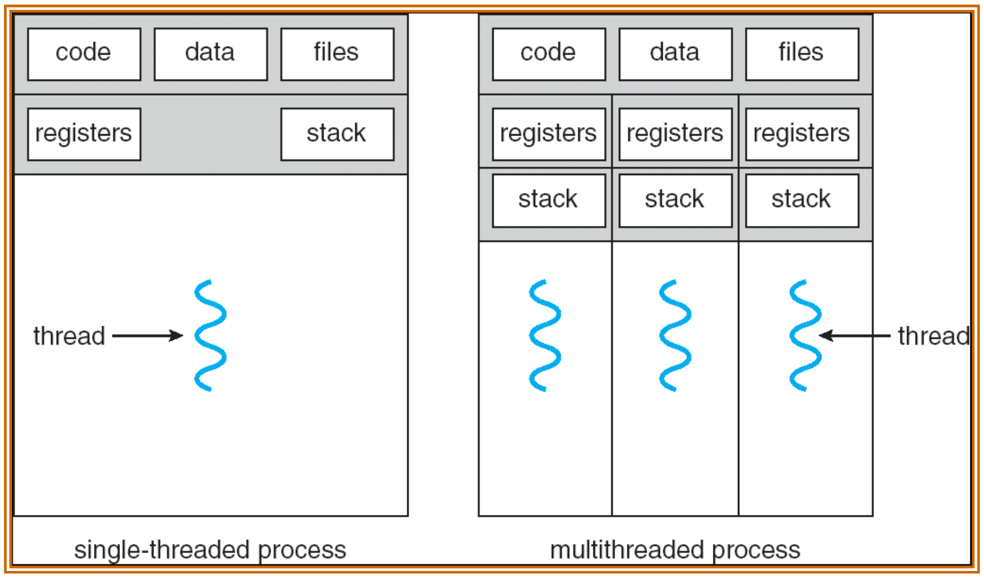
- ❖ **Responsiveness**: may allow continued execution if part of process is blocked, especially important for user interfaces
- ❖ **Resource Sharing**: threads share resources of process, easier than shared memory or message passing
- ❖ **Economy**: cheaper than process creation, thread switching lower overhead than context switching
- ❖ **Scalability**: process can take advantage of multiprocessor architectures

Multicore Programming (multiprocessor):

- ❖ Dividing activities
- ❖ Balance
- ❖ Data splitting
- ❖ Data dependency
- ❖ Testing and debugging

- **Parallelism** **(multi-core)** implies a system can perform more than one task simultaneously
  - **Data parallelism** – distributes subsets of the same data across multiple cores, same operation on each
  - **Task parallelism** – distributing threads across cores, each thread performing unique operation
- **Concurrency** **(single-core)** supports more than one task making progress, Single processor / core, scheduler providing concurrency

Single and Multithreaded Processes



Multithreading Models

- **Many-to-One**
  - Many user-level threads mapped to single kernel thread
  - One thread blocking causes all to block
  - Multiple threads may not run in parallel on muticore system because only one may be in kernel at a time
- **One-to-One**
  - Each user-level thread maps to kernel thread
  - Creating a user-level thread creates a kernel thread
  - More concurrency than many-to-one
  - Number of threads per process sometimes restricted due to overhead
- **Many-to-Many**
  - Allows many user level threads to be mapped to many kernel threads
  - Allows the operating system to create enough kernel threads
- **Two-level Model:** allows a user thread to be bound to kernel thread

Threading Issues

- Semantics of **fork ()** and **exec ()** system calls
- Signal handling
  - Synchronous and asynchronous
- Thread cancellation of target thread
  - Asynchronous or deferred
  - Thread-local storage
  - Scheduler Activations

Chapter 5

Dispatcher

- gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- **Dispatch latency** – time it takes for the dispatcher to stop one process and start another running

Scheduling 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
- **Waiting time**: amount of time a process has been waiting in the ready queue
- **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)

First-Come, First-Served (FCFS) Scheduling **(read from slides)**

- فكرته: المهمة التي تأتي أولا تنفذ أولا وبشكل كامل أي أنها تبقى في المعالج حتى تنتهي
- عيوبه: متوسط وقت الانتظار يعتمد بشكل كبير على ترتيب وصول المهام وكذلك أن مهمة قصيرة قد تنتظر مهمة طويلة في أنظمة الـ (time sharing)
- مميزاته: بسيط

- **Convoy effect** - short process behind long process

Shortest-Job-First (SJF) Scheduling **(read from slides)**

- فكرته: أن المهمة التي لا تستغرق وقت طويل في المعالج تدخل اولا وتنقسم هذه الطريقة الى نوعان:
  - النوع الاول: (**nonpreemptive**): عند قدوم عدد من المهام الى المعالج يأخذ الـمهمة التي لأتأخذ وقت طويل ولكن عند قدوم مهمة الوقت اللازم لها اقل من التي مع المعالج فانه يتم تجاهلها ويكمل المعالج عمله الى ان ينتهي
  - النوع الثاني: (**preemptive**): نفس الطريقة السابقة ولكن الفرق هو عند قدوم مهمة الوقت اللازم لها اقل من التي مع المعالج فان المعالج يوقف العملية ويضع المهمة التي معها في (queue ready)ويأخذ المهمة القادمة و هكذا
- عيوبه: صعوبة برمجته وذلك لعدم معرفة الوقت اللازم لبقاء المهمة الجديدة في المعالج.
- مميزاته: مثالي

- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time
- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request
  - Could ask the user

Priority Scheduling (read from slides)

- ❖ فكرته: يعتمد على (**A priority number**) وهو عباره عن رقم صحيح يأتي مع الـمهمة يمثل الاولوية أي ان المهمة التي معها عدد صغير له اولويه أكبر (في نظام يونكس مثلا) عن مهمة التي معها رقم كبير أي أن لها الحق في الدخول للمعالج
- ❖ عيوبه: التجويع (**Starvation**)هو ان الأقل أولوية قد لا تنفذ ابدا لذلك الحل هو تقليل الرقم الذي يمثل الـمهمة من فتره الى اخرى الى ان يصبح ذو اولويه عليا فينفذ (هذا بالنسبة للمهمة التي ليست لها اولويه عليا وذلك حتى لا تظل بدون تنفيذ)

- ❖ A priority number (integer) is associated with each process
- ❖ The CPU is allocated to the process with the **highest priority** (smallest integer ≡ highest priority)
  - o **Preemptive**
  - o **Nonpreemptive**
  - o SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- ❖ Problem ≡ **Starvation** – low priority processes may never execute
- ❖ Solution ≡ **Aging** – as time progresses increase the priority of the process

Round Robin (RR) (read from slides)

فكرته: الطريقة هذه لا تنظر لأهمية الـمهمة جميعهم متساوون. يحدد وقت للمعالج ولنقل 20 عادة يكون 10 - 100 millisecondsومعنى ذلك ان المهمة سواء كانت لها أولوية او لا تأخذ 20 من الزمن فقط وتخرج من المعالج حتى ولو لم تنتهي إذا انتهت لا توجد مشكلة اما اذا لم تنتهي فإنها تخرج وتوضع في اخر الطابور

- ❖ Each process gets a small unit of CPU time (time quantum q), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- ❖ If there are n processes in the ready queue and the time quantum is q, then each process gets 1/n of the CPU time in chunks of at most q time units at once. No process waits more than (n-1)q time units.
- ❖ Timer interrupts every quantum to schedule next process
- ❖ Performance
  - o q large ⇒ FIFO
  - o q small ⇒ q must be large with respect to context switch, otherwise overhead is too high

Thread Scheduling

- ❖ Distinction between user-level and kernel-level threads
- ❖ When threads supported, threads scheduled, not processes
- ❖ Many-to-one and many-to-many models, thread library schedules user-level threads to run on LWP
  - o Known as process-contention scope (PCS) since scheduling competition is within the process
  - o Typically done via priority set by programmer
- ❖ Kernel thread scheduled onto available CPU is system-contention scope (SCS) – competition among all threads in system

Multilevel Queue

- Ready queue is partitioned into separate queues, eg:
    - foreground (interactive)
    - background (batch)
- Process permanently in a given queue
    - Each queue has its own scheduling algorithm:
    - foreground – RR
    - background – FCFS
    - Scheduling must be done between the queues:
    - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
    - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR
    - 20% to background in FCFS
- Multilevel-feedback-queue scheduler defined by the following parameters:
    - number of queues
    - scheduling algorithms for each queue
    - method used to determine when to upgrade a process
    - method used to determine when to demote a process
    - method used to determine which queue a process will enter when that process needs service

Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available
- Homogeneous processors within a multiprocessor
- Asymmetric multiprocessing – only one processor accesses the system data structures, alleviating 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
    - Currently, most common
- Processor affinity – process has affinity for processor on which it is currently running
    - soft affinity
    - hard affinity
    - Variations including processor sets

Multiple-Processor Scheduling – Load Balancing

- If SMP, need to keep all CPUs loaded for efficiency
- Load balancing attempts to keep workload evenly distributed
- 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

Race Condition

- Because of the timing and which process starts first
- There is a chance that *different executions may end up with different results*

Critical Section Problem

- Each process has critical section segment of code
  - Process may be changing common variables, updating table, writing file.
  - When one process in critical section, no other may be in its critical section
  - Critical section problem is to design protocol to solve this
- Each process must ask permission to enter critical section in entry section, may follow critical section with exit section, then remainder section

Solution to Critical-Section Problem

- Mutual Exclusion - If Process Pi is executing in its critical section, then no other processes can be executing in their critical sections
- 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
- Bounded Waiting - A bound must exist on the number of times that other processes can enter their critical sections after a process has made a request to enter its critical section and before that request is granted
  - Assume that each process executes at a nonzero speed
  - No assumption concerning relative speed of the n processes
- Two approaches depending on if kernel is preemptive or non-preemptive
  - Preemptive – allows preemption of process when running in kernel mode
  - Non-preemptive – runs until exits kernel mode, blocks, or voluntarily yields CPU
    - Essentially free of race conditions in kernel mode

Mutex Locks

- Previous solutions are complicated and generally inaccessible to application programmers
- OS designers build software tools to solve critical section problem
- Simplest is Mutex lock
- Product critical regions with it by first acquire **()** a lock then release **()** it
  - Boolean variable indicating if lock is available or not
- Calls to acquire **()** and release **()** must be atomic
  - Usually implemented via hardware atomic instructions
- But this solution requires busy waiting
  - This lock therefore called a spinlock

Semaphore

- Synchronization tool that does not require busy waiting
- Semaphore ***S*** – integer variable
- Two standard operations modify ***S***: wait **()** and signal ()
  - Originally called P () and V ()
- Less complicated
- Can only be accessed via two indivisible (atomic) operations

Classical Problems of Synchronization

- Classical problems used to test newly-proposed synchronization schemes
  - Bounded-Buffer Problem
  - Readers and Writers Problem
  - Dining-Philosophers Problem

Bounded-Buffer Problem

- ***n*** buffers, each can hold one item
- Semaphore **Mutex** initialized to the value 1
- Semaphore **full** initialized to the value 0
- Semaphore **empty** initialized to the value n

Readers and Writers Problem

- A data set is shared among a number of concurrent processes
  - Readers – only read the data set; they do ***not*** perform any updates
  - Writers – can both read and write
- Problem – allow multiple readers to read at the same time
  - Only one single writer can access the shared data at the same time
  - Several variations of how readers and writers are treated – all involve priorities
- Shared Data
  - Data set
  - Semaphore **rw_mutex** initialized to 1
  - Semaphore **Mutex** initialized to 1
  - Integer **read_count** initialized to 0
- **Readers-Writers Problem Variations**
  - *First* variation – no reader kept waiting unless writer has permission to use shared object
  - *Second* variation – once writer is ready, it performs write asap
  - Both may have starvation leading to even more variations
  - Problem is solved on some systems by kernel providing reader-writer locks

Dining-Philosophers Problem

- Philosophers spend their lives thinking and eating
- Don't interact with their neighbors, occasionally try to pick up 2 chopsticks (one at a time) to eat from bowl
  - Need both to eat, then release both when done
- In the case of 5 philosophers
  - Shared data
    - Bowl of rice (data set)
    - Semaphore chopstick [5] initialized to 1

Chapter 7

Deadlock Characterization

- 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 {P0, P1, …, Pn} of waiting processes such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2, …, Pn–1 is waiting for a resource that is held by Pn, and Pn is waiting for a resource that is held by P0.

Resource-Allocation Graph **(read from slides)**

- A set of vertices V and a set of edges E.
- V is partitioned into two types:
  - P = {P1, P2, …, Pn}, the set consisting of all the processes in the system
  - R = {R1, R2, …, Rm}, the set consisting of all resource types in the system
- request edge: directed edge Pi $\rightarrow$ Rj
- assignment edge: directed edge Rj $\rightarrow$ Pi

Basic Facts

- If graph contains no cycles $\Rightarrow$ no deadlock
- If graph contains a cycle $\Rightarrow$
  - if only one instance per resource type, then deadlock
  - if several instances per resource type, possibility of deadlock

Methods for Handling Deadlocks

- Ensure that the system will never enter a deadlock state
- Allow the system to enter a deadlock state and then recover
- Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX

Deadlock Prevention

- Mutual Exclusion – not required for sharable resources; must hold for non-sharable resources
- Hold and Wait – must guarantee that whenever a process requests a resource, it does not hold any other resources
  - Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none
  - Low resource utilization; starvation possible
- No Preemption:
  - If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released
  - Preempted resources are added to the list of resources for which the process is waiting
  - Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting
- Circular Wait: impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration

Safe State

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state
- System is in safe state if there exists a sequence <P1, P2, …, Pn> of ALL the processes in the systems such that for each Pi, the resources that Pi can still request can be satisfied by currently available resources + resources held by all the Pj, with j < I
- That is:
  - If Pi resource needs are not immediately available, then Pi can wait until all Pj have finished
  - When Pj is finished, Pi can obtain needed resources, execute, return allocated resources, and terminate
  - When Pi terminates, Pi +1 can obtain its needed resources, and so on

Basic Facts

- If a system is in safe state, ⇒ no deadlocks
- If a system is in unsafe state ⇒ possibility of deadlock
- Avoidance ⇒ ensure that a system will never enter an unsafe state

Avoidance algorithms

- Single instance of a resource type
  - Use a resource-allocation graph
- Multiple instances of a resource type
  - Use the banker's algorithm

Recovery from Deadlock: Process Termination

- ❖ Abort all deadlocked processes
- ❖ Abort one process at a time until the deadlock cycle is eliminated
- ❖ In which order, should we choose to abort?
  1. Priority of the process
  2. How long process has computed, and how much longer to completion
  3. Resources the process has used
  4. Resources process needs to complete
  5. How many processes will need to be terminated
  6. Is process interactive or batch?

Recovery from Deadlock: Resource Preemption

- ❖ Selecting a victim – minimize cost
- ❖ Rollback – return to some safe state, restart process for that state
- ❖ Starvation – same process may always be picked as victim, include number of rollback in cost factor

Chapter 8

Binding of Instructions and Data to Memory

- Address binding of instructions and data to memory addresses can happen at three different stages
  - Compile time: If memory location known a priori, absolute code can be generated; must recompile code if starting location changes
  - Load time: Must generate relocatable code if memory location is not known at compile time
  - Execution time: Binding delayed until run time if the process can be moved during its execution from one memory segment to another
    - Need hardware support for address maps (e.g., base and limit registers)

Logical vs. Physical Address Space

- The concept of a logical address space that is bound to a separate physical address space is central to proper memory management
  - Logical address – generated by the CPU; also referred to as virtual address
  - Physical address – address seen by the memory unit
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme
- Logical address space is the set of all logical addresses generated by a program
- Physical address space is the set of all physical addresses generated by a program

Dynamic relocation using a relocation register

- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded
- All routines kept on disk in relocatable load format
- Useful when large amounts of code are needed to handle infrequently occurring cases
- No special support from the operating system is required
  - Implemented through program design
  - OS can help by providing libraries to implement dynamic loading

Dynamic Linking

- Static linking – system libraries and program code combined by the loader into the binary program image
- Dynamic linking –linking postponed until execution time
- Small piece of code, stub, used to locate the appropriate memory-resident library routine
- Stub replaces itself with the address of the routine, and executes the routine
- Operating system checks if routine is in processes' memory address
    - If not in address space, add to address space
- Dynamic linking is particularly useful for libraries
- System also known as shared libraries
- Consider applicability to patching system libraries
    - Versioning may be needed

Swapping

- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution
    - Total physical memory space of processes can exceed physical memory
- Backing store – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images
- Roll out, roll in – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped
- System maintains a ready queue of ready-to-run processes which have memory images on disk
- Does the swapped-out process need to swap back in to same physical addresses?
- Depends on address binding method
    - Plus, consider pending I/O to / from process memory space
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)
    - Swapping normally disabled
    - Started if more than threshold amount of memory allocated
    - Disabled again once memory demand reduced below threshold

Contiguous Allocation

- Main memory usually into two partitions:
    - Resident operating system, usually held in low memory with interrupt vector
    - User processes then held in high memory
- Relocation registers used to protect user processes from each other, and from changing operating-system code and data
    - Base register contains value of smallest physical address
    - Limit register contains range of logical addresses – each logical address must be less than the limit register
    - MMU maps logical address *dynamically*
- Multiple-partition allocation
    - Variable-partition sizes for efficiency (sized to a given process' needs)
    - Hole – block of available memory; holes of various size are scattered throughout memory
    - When a process arrives, it is allocated memory from a hole large enough to accommodate it
    - Operating system maintains information about:
      a) allocated partitions b) free partitions (hole)

Dynamic Storage-Allocation Problem

- First-fit: Allocate the first hole that is big enough
- Best-fit: Allocate the smallest hole that is big enough; must search entire list, unless ordered by size
    - Produces the smallest leftover hole
- Worst-fit: Allocate the largest hole; must also search entire list
    - Produces the largest leftover hole

Fragmentation

- External Fragmentation – total memory space exists to satisfy a request, but it is not contiguous
- Internal Fragmentation – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used

Reduce external fragmentation by compaction

- Shuffle memory contents to place all free memory together in one large block
- Compaction is possible *only* if relocation is dynamic, and is done at execution time
- I/O problem
    - Latch job in memory while it is involved in I/O
    - Do I/O only into OS buffers

Segmentation

- Memory-management scheme that supports user view of memory
- A program is a collection of segments
    - A segment is a logical unit such as: main program, procedure, function, method, object, local variables, global variables, common block, stack, symbol table, arrays.

Segmentation Architecture

- Logical address consists of a two tuple: <segment-number, offset>,
- Segment table – maps two-dimensional physical addresses; each table entry has:
  - base – contains the starting physical address where the segments reside in memory
  - limit – specifies the length of the segment
- Segment-table base register (STBR) points to the segment table's location in memory
- Segment-table length register (STLR) indicates number of segments used by a program;

  segment number s is legal if s < STLR

Paging

- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
  - Avoids external fragmentation
  - Avoids problem of varying sized memory chunks
- Divide physical memory into fixed-sized blocks called frames
  - Size is power of 2, between 512 bytes and 16 Mbytes
- Divide logical memory into blocks of same size called pages
- Keep track of all free frames
- To run a program of size ***N*** pages, need to find ***N*** free frames and load program
- Set up a page table to translate logical to physical addresses
- Backing store likewise split into pages
- Still have Internal fragmentation

Important Question:

1-Describe the dining-philosopher’s problem and how it relates to operating systems.

The scenario involves five philosophers sitting at a round table with a bowl of food and five chopsticks. Each chopstick sits between two adjacent philosophers. The philosophers are allowed to think and eat. Since two chopsticks are required for each philosopher to eat, and only five chopsticks exist at the table, no two adjacent philosophers may be eating at the same time. A scheduling problem arises as to who gets to eat at what time. This problem is similar to the problem of scheduling processes that require a limited number of resources.

2- Explain the basic method for implementing paging.
Physical memory is broken up into fixed-sized blocks called frames while logical memory is broken up into equal-sized blocks called pages. Whenever the CPU generates a logical address, the page number and offset into that page is used, in conjunction with a page table, to map the request to a location in physical memory.

3- Give Three examples of recourse type:
CPU cycles, memory space, I/O devices

4- Explain the sequence of events that happens when a page-fault occurs
When the operating system cannot load the desired page into memory, a pagefault occurs. First, the memory reference is checked for validity. In the case of an invalid request, the program will be terminated. If the request was valid, a free frame is located. A disk operation is then scheduled to read the page into the frame just found, update the page table, restart the instruction that was interrupted because of the page fault, and use the page accordingly.

5-What are the differences between internal and external fragmentation?
Fragmentation occurs when memory is allocated and returned to the system. As this occurs, free memory is broken up into small chunks, often too small to be useful. External fragmentation occurs when there is sufficient total free memory to satisfy a memory request, yet the memory is not contiguous, so it cannot be assigned. Some contiguous allocation schemes may assign a process more memory than it actually requested. Internal fragmentation occurs when a process is assigned more memory than it has requested and the wasted memory fragment is internal to a process.