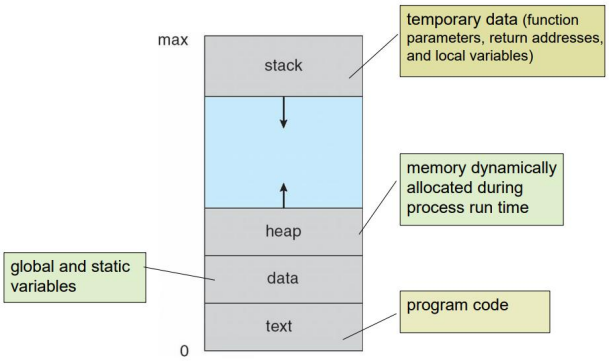
Lecture1 Processes

1. Process concept

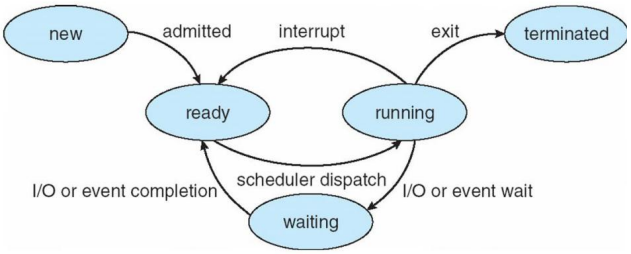
• Definition: a program in execution (program becomes process when executable file loaded into memory)

• An operating system executes a variety of programs: Batch system-jobs, Time-shared system-user programs or tasks

• progress in memory



• process state(defined by the current activity of that process): new, running, waiting, ready, terminated



• **Process Control Block(PCB)**

-PCB, also called as context of the process, is a data structure used for storing the information about a process

-Each &every process is identified by its own PCB. PCB of each process resides in the main memory. PCB of all the processes are present in a linked list.

2. Features of process

(1) Scheduling:

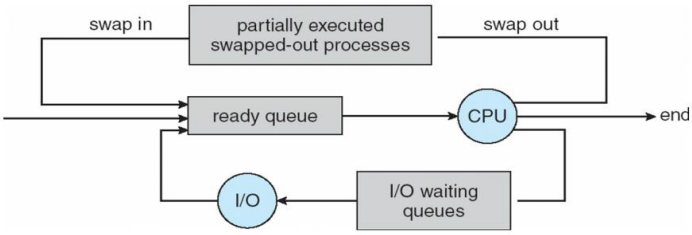
• maintains scheduling queues of processes: job queue, ready queue, device queue

• schedulers:

a. Long-term scheduler(Job scheduler): new

b. Short-term scheduler(CPU scheduler): ready-->running

c. Medium-term scheduler: **swapping**(remove process from memory, store on disk, bring back in from disk to continue execution



• representation of progress scheduling: **queueing diagram** represents queues, resources, flows

• **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.

(2) Creation: Parent process create children processes, which, in turn create other processes, forming a tree of processes

• Resource sharing options:

--Parent and children share all resources

--Children share subset of parent's resources

--Parent and child share no resources

• Execution options:

--Parent and children execute concurrently

--Parent waits until children terminate

(3) Termination

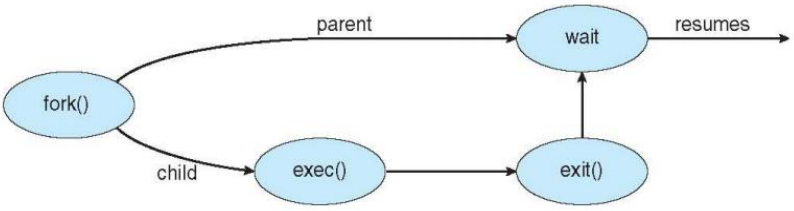
• Process executes last statement and then asks the operating system to delete it using the exit() system call.

--Returns status data from child to parent (via wait())

--process' resources are deallocated by operating system

• Parent may terminate the execution of children processes.

--Child has exceeded allocated resources



3. Interprocess communication

• Independent Processes

Cooperating Processes: can effect or be affected by other process.

• communication Models

① Message passing: communication takes place by way of messages exchanged among the cooperating processes.(The message size is either fixed or variable)-->send(message), receive(message)

a. Direct or indirect communication:

• exchanged messages by communicating processes reside in a temporary queue

- buffering: zero capacity, bounded capacity, unbounded capacity

i. Direct communication: processes must name each other explicitly

- send(P,message): send a message to P

-receive(Q,message): receive a message from process Q

ii. Indirect communication: create a new **mailbox**(port), send and receive messages through mailbox, destroy a mailbox

-send(A,message): send a message to mailbox A

-receive(A,message): receive a message from mailbox A

b. Synchronous or asynchronous communication

- Blocking is considered synchronous

-Blocking send: the sender is blocked until the message is received

-Blocking receive: the receiver is blocked until a message is available

- Non-blocking is considered asynchronous

-Non-blocking send: the sender sends the message and continue

-Non-blocking receive: the receiver receives: A valid message, or Null message

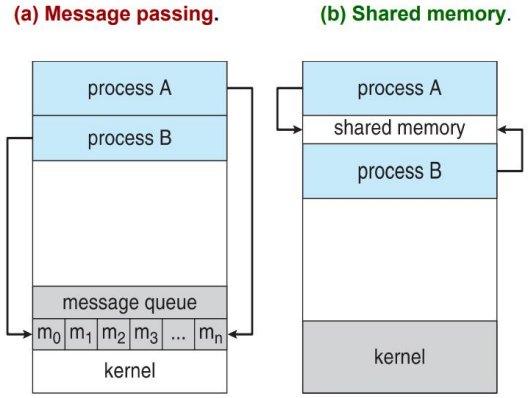
c. Automatic or explicit buffering

② Shared memory: a region of memory is shared by cooperating process

-->two types of buffers can be used:

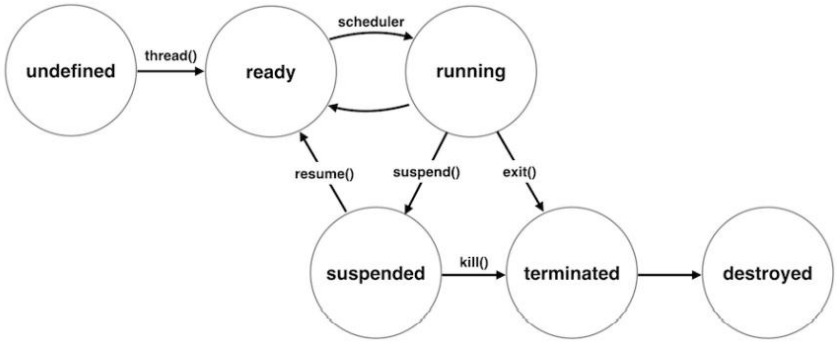
a) Unbounded-buffer places no practical limit on the size of the buffer

b) Bounded-buffer assumes that there is a fixed buffer size



Lecture2 Threads

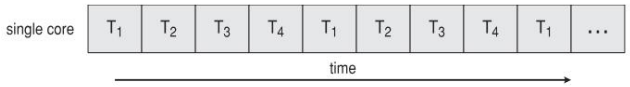
1. Overview
   - A thread is the smallest unit of processing that can be performed in an OS
   - a thread is a lightweight process that can be managed independently by a scheduler. Executes a series of instructions in order (only one thing happens at a time).
   - When there are multiple threads running for a process, the process provides common memory. Multiple tasks with the application can be implemented by separate threads
   - composition:
     ① A thread ID
     ② A program counter: keeps the track of instructions and tells which instruction to execute next.
     ③ A register set: keeps the track of the current working variable of a thread.
     ④ A stack: contains the history of thread execution
     - Thread states: undefined state, ready state, running state, suspended state, terminated state



   - single and multithreaded process

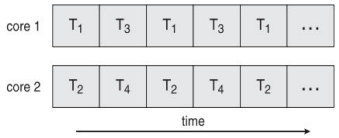
2. Multicore programming
   - concurrency vs. Parallelism
     ① Concurrent execution on single-core system:

     Concurrency means multiple tasks which start, run, and complete in overlapping time periods, in no specific order.



     ② Parallelism on a multi-core system:

     A system is parallel if it can perform more than one task simultaneously.



   - **Amdahl's Law**

     --An application may process the task serially from start to end (one task at the

time) or split the task up into subtasks which can be completed in parallel.

--Amdahl's law can be used to calculate (theoretically) how much a computation can be speed up by running part of it in parallel using multiple processors

S is serial portion of the application that must be performed serially on a system with
N - processing cores

$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

3. Multithreading models
   - Types of threads
     ① User Threads: is the unit of execution that is implemented by users and the kernel is not aware of the existence of these threads.
     -->User-level threads are much faster than kernel level threads
     -->All thread management is done by the application by using a thread library.
     ② Kernel Threads: is the unit of execution that is scheduled by the kernel to execute on the CPU
     -->is the unit of execution that is scheduled by the kernel to execute on the CPU

User Level Threads	Kernel Level Thread
User level threads are faster to create and manage.	Kernel level threads are slower to create and manage.
Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads
User level thread is generic and can run on any operating system	Kernel level thread is specific to the operating system.
Multi-threaded application cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.

   - Multithreading models: threads-kernel
     ① Many-to-one
     --The process can only run one user-level thread at a time because there is only one kernel-level thread associated with the process.
     --If any user thread is blocked due to some reason, the entire process will get blocked
     ② One-to-one
     --This model provides more concurrency than the Many-to-One model. If one thread makes a blocking call, it allows another thread to run.
     ③ Many-to-many
     --If one or more user threads makes blocking call, so, other kernel threads will manage the blocking issue and continue executing the task.
     --Allows the operating system to create a sufficient number of kernel threads

4. Thread libraries
   - Definition: **Thread library** is an **Application Programming Interface** (**API**) for

programmer to create and manage threads in their applications

- two approaches for implementing thread library:
  ① to provide a library entirely in user space with no kernel support. This means that invoking a function in the library results in a local function call in user space and not a system call.
  ② to implement a kernel-level library supported directly by the operating system. Code and data structures for the library exist in kernel space. Invoking a function in the API for the library typically results in a system call to the kernel.
- Thread creation
  ① Asynchronous - Parent thread creates child then executes independently. It means, little data sharing between parent and child thread.
  ② Synchronous - Parent thread waits for the child thread to finish its execution. More of data sharing is done here.

5. Implicit threading

- The category where the programmer creates and manages threads is an **explicit threading**.

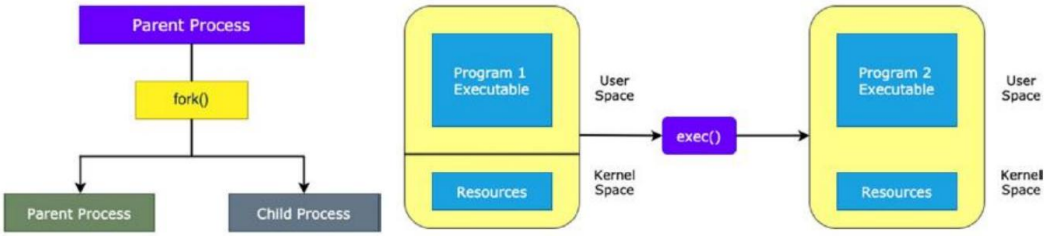
The category where the compilers and run-time libraries create and manage threads is an **implicit threading**.

-->Three alternative approaches for designing multithreaded programs:

① Thread pool - to create a number of threads at process startup and place them into a pool, where they sit and wait for work. Threads are allocated from pool as needed, and return to the pool when no longer needed.
② OpenMP is a set of compiler directives available for C, C++, and Fortran programs that instruct the compiler to automatically generate parallel code where appropriate.
③ Grand Central Dispatch (GCD) - is an extension to C and C++ available on Apple's MacOS X and iOS operating systems to support parallelism.

6. Threading issues / designing multithreaded programs

(1) Semantics of **fork()** and **exec()** system calls



(2) Signal handling:

- A signal is a software interrupt or an event generated by a Unix/Linux system in response to a condition or an action.
- The signal is handled by a signal handler(all signals are handled exactly once)
  a. asynchronous signal is one that is generated from outside the process that receives it
  b. synchronous signal is delivered to the same process that caused the

signal to occur

(3) Thread cancellation

- Thread cancellation is the act of terminating a thread before it has completed
- A thread that is to be canceled is often referred to as the **target thread**.
- Threads can be cancelled in a couple of ways:
    a. Asynchronous cancellation terminates the target thread immediately
    --> Thread may be in the middle of writing data ... not so good
    b. Deferred cancellation allows the target thread to periodically check if it should be cancelled
    --> Allows thread to terminate itself in an orderly fashion

Lecture3 Process Synchronization

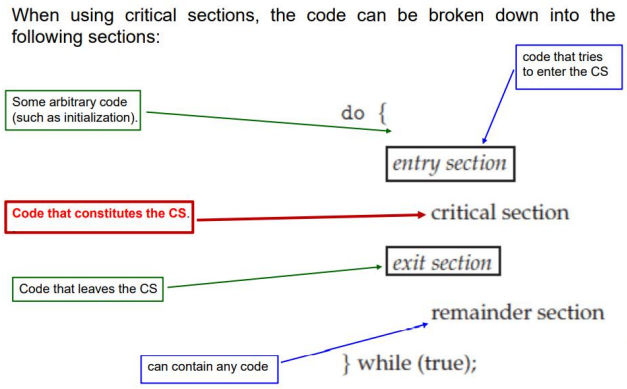
1. Background

   • definition: **PS** is the task of coordinating the execution of processes in a way that no two processes can have access to the same shared data and resources.

   • where multiple processes access and manipulate the same data concurrently-->Data inconsistency can result in what is called a **race condition**.-->To prevent race conditions, concurrent processes must be synchronized

   Process Synchronization is a way to coordinate processes that use shared data.--> Basic idea in synchronization: need locks in one form or another.

2. The critical-section problem



   • A sequence of code that operates on shared state is a **critical section**.

   • allocation of shared objects:

   ① can be dynamically allocated from the heap -->each thread that uses an object needs a pointer or reference to it

   ② can be statically allocated in global memory by declaring static variables in the program.-->then a thread's code can just refer to an objects global name to reference it; the compiler computes the corresponding address.

   • solutions to CS problem

   ① Mutual exclusion: No more than one process can execute in its critical section at one time.

   ② Progress: If no one is in the critical section and someone wants in, then those processes not in their remainder section must be able to decide in a finite time who should go in

   ③ Bounded waiting: No process should have to wait forever to enter into the critical section

   • Two general approaches:

   ① A preemptive kernel allows a process to be preempted while it is running in kernel mode

   ② A non-preemptive kernel does not allow a process running in kernel mode to be preempted

   • design mechanisms that allow a single thread to be in its CS at one time: Peterson's

solution, Synchronization hardware, Mutex locks/mutual exclusion, Semaphores

3. Peterson's solution: restricted to two processes/threads that alternate execution between their critical sections and remainder sections.
   ① int turn; // to indicate the process whose turn is to enter the CS
   ② boolean flag[2]; // Initialized to FALSE, initially no one is interested in entering the critical section

   -->each Pi enters its critical section only if either: flag[j] = false or turn = i

```
do {
    flag[i] = TRUE;
    turn = j;
    while ( flag[j] && turn == j);

        CRITICAL SECTION

    flag[i] = FALSE;

        REMAINDER SECTION
}
while (TRUE);
```

4. Synchronization hardware: rely on some special machine instructions: locking
   ① Single-processor environment -**Test and Set solution**

```
do {
   while (test_and_set(&lock))
      ; /* do nothing */

      /* critical section */

   lock = false;

      /* remainder section */
} while (true);
```

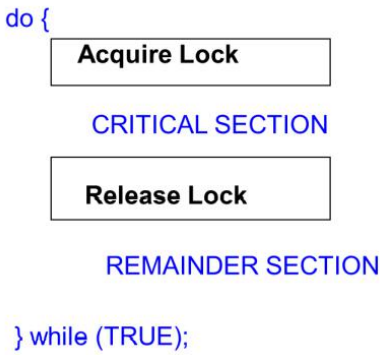
   ② Multi-processor environment -**Compare and Swap** solution

   - a global variable (lock) is declared and is initialized to 0.
   - the first process that invokes *compare and swap()* will set lock to 1.
   - it will then enter its critical section, because the original value of lock was equal to the expected value of 0.

```
do {
   while (compare_and_swap(&lock, 0, 1) != 0)
      ; /* do nothing */

      /* critical section */

   lock = 0;

      /* remainder section */
} while (true);
```

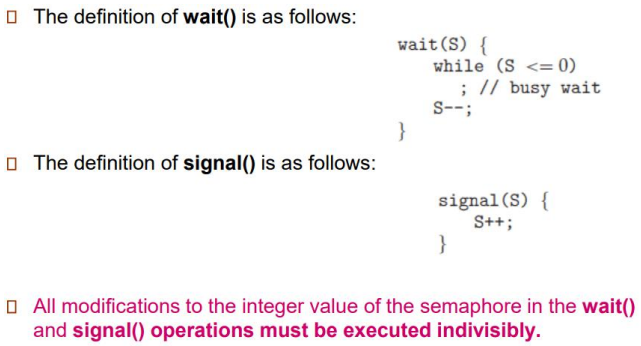
5. Mutex locks/mutual exclusion: software to protect critical regions and avoid race conditions
   ① To enforce mutex at the kernel level and prevent the corruption of shared data structures --> disable interrupts for the smallest number of instructions is the best way.
   ② To enforce mutex in the software areas – use the **busy-wait** mechanism

(busy-wait mechanism or busy-looping or spinning is a technique in which a process/thread repeatedly checks to see if a lock is available.)



--> disadvantage: require busy waiting(This type of mutex lock is a **spinlock** because the process "spins" while waiting for the lock to become available.)

6. Semaphores: is simply a variable which is non-negative and shared between threads.(**wait(), signal()**)



- two main types of semaphores

③ COUNTING SEMAPHORE – allow an arbitrary resource count. It is used to control access to a resource that has multiple instances.

--The semaphore S is initialized to the number of resources available.

--Each process that wishes to use a resource performs a **wait()** operation on the semaphore (thereby decrementing the count).

--When a process releases a resource, it performs a **signal()** operation (incrementing the count).

--When the count for the semaphore goes to 0, all resources are being used. After that, processes that wish to use a resource will block until the count becomes greater than 0.

④ BINARY SEMAPHORE – This is also known as mutex lock. It can have only two values – 0 and 1. Its value is initialized to 1.

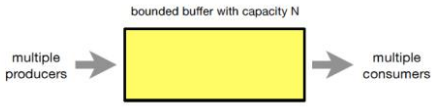
--The wait() operation only works when the semaphore is 1 and the signal() operation succeeds when semaphore is 0.

-->disadvantage: **starvation** & **deadlock**

7. Classical problems of synchronization

(2) The Bounded-Buffer Problem / The Producer-Consumer Problem

```
int n;
semaphore mutex = 1;
semaphore empty = n;
semaphore full = 0
```



(3) The Readers – Writers Problem

A data set is shared among a number of concurrent processes.

• Only one single writer can access the shared data at the same time, any other writers or readers must be blocked.

• Allow multiple readers to read at the same time, any writers must be blocked.

--> Solution: Acquiring a reader – writer lock requires specifying the mode of the lock: either read or write access.

(4) The Dining-Philosophers Problem: 5 philosophers + 5 chopsticks --> several solutions to the deadlock problem

① Allow at most four philosophers to be sitting simultaneously at the table.

② Allow a philosopher to pick up her chopsticks only if both chopsticks are available

③ Use an asymmetric solution:

• an odd-numbered philosopher picks up first her left chopstick and then her right chopstick

• an even numbered philosopher picks up her right chopstick and then her left chopstick

Lecture4　CPU Scheduling I

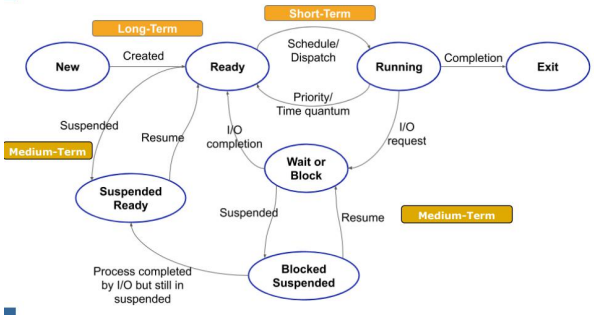
1. Basic concepts

- **CPU - I/O Burst Cycle**
- **The CPU scheduler** is the mechanism to select which process has to be executed next from a ready list(also known as run queue) and allocates the CPU to that process.
- Types of schedulers:

① Long-Term Scheduler

② Short-Term Scheduler

③ Medium-Term Scheduler



- Preemptive scheduling & non-preemptive scheduling
- CPU scheduling takes place on 4 circumstances

① When the process changes state from Running to Ready.-->preemptive

② Changes state from Running to Waiting.-->nonpreemptive

③ Changes state from Waiting to Ready. -->preemptive

④ Process Terminates.-->nonpreemptive

- Dispatcher

1) **Dispatcher module** 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

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

2. scheduling criteria: max CPU utilization, max throughput, min turnaround time, min waiting time, min response time

Arrival Time: *Time at which the process arrives in the ready queue.*
Completion Time: *Time at which process completes its execution.*
Burst Time: *Time required by a process for CPU execution.*
Turnaround Time: *the total amount of time spent by the process from coming in the ready state for the first time to its completion.*

Turnaround time = Exit time - Arrival timer.

Waiting Time (W.T): *The total time spent by the process/thread in the ready state waiting for CPU.*

Waiting Time = Turn Around Time – Burst Time

Response time: *Time at which the process gets the CPU for the first time.*

Response time = Time at which the process gets the CPU for the first time - Arrival time

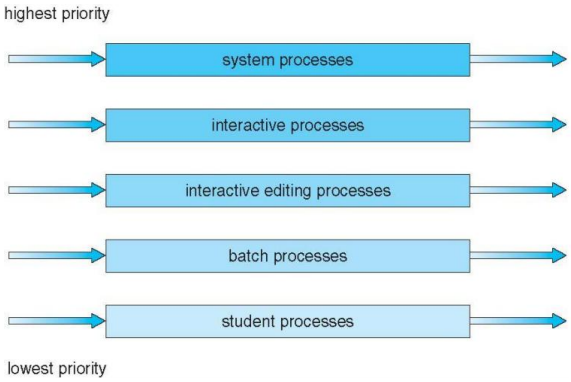
3. scheduling algorithms
   (1) First-Come, First-Served (FCFS) Scheduling
   (2) Shortest-Job-First (SJF) Scheduling (non-preemptive scheduling)
   -->advantage: gives minimum average waiting time for a given set of processes
   -->difficulty: knowing the length of the next CPU request
   a) For long-term scheduling in a batch system, we can use the process time limit that a user specifies when he submits the job.
   b) with short-term scheduling, we can only Computing an approximation of the length of the next CPU burst

   Generally predicted as an exponential average of the measured lengths of previous CPU bursts.

   Let t_n = actual length of the n^{th} burst;
   τ_n = the predicted value for the next of the n^{th} CPU burst;
   a = a weighing factor, $0 \le a \le 1$.
   The estimate of the next CPU burst period is:

$$\tau_{n+1} = at_n + (1-a)\tau_n$$

   (3) Shortest Remaining Time First(preemptive scheduling)
   (4) Priority Scheduling
   • Priorities may be:
   a. **Internal priorities** are determined by the system using factors such as time limits, a process' memory requirements, and any other system-related factors.
   b. **External priorities** are assigned by administrators.
   -->problem: Starvation: low priority processes may never execute
   -->SOLUTION: Aging: as time progresses increase the priority of the process
   (5) Round Robin(RR) Scheduling
   • Each process gets a small unit of CPU time (**time quantum** or **time-slice**), usually 10-100 milliseconds.
   • After this time has elapsed, the process is preempted and added to the end of the ready queue
   (6) Multiple-Level Queues Scheduling
   • We can group processes into priority classes and assign a separate run queue for each class.
   • Processes can not move between queues.



(7) Multilevel Feedback Queue Scheduling

• Multilevel feedback queues - automatically place processes into priority levels based on their CPU burst behavior.: I/O-intensive processes will end up on higher priority queues and CPU-intensive processes will end up on low priority queues.

• A process can move between the various queues

• A multilevel feedback queue uses two basic rules:

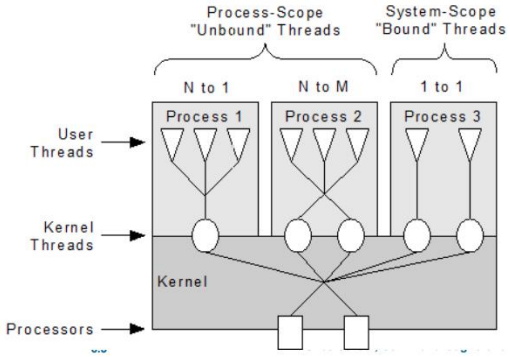
a. A new process gets placed in the highest priority queue.

b. If a process does not finish its quantum then it will stay at the same priority level otherwise it moves to the next lower priority level

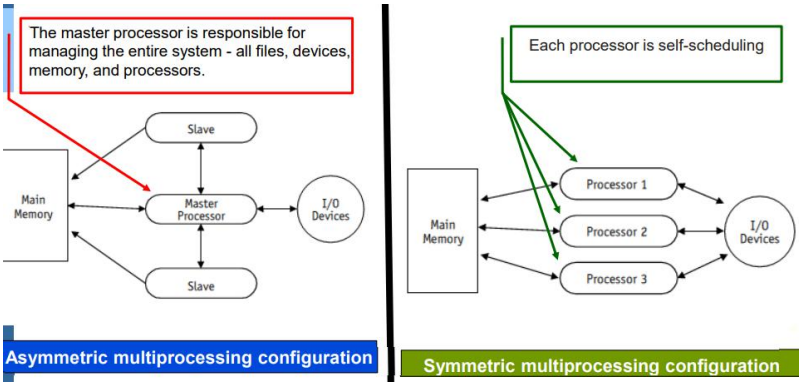
• Multilevel-feedback-queue scheduler defined by the following parameters: number of queues, scheduling algorithms for each queue, method used to determine when to upgrade/demote a process, method used to determine which queue a process will enter when that process needs service

Lecture5 CPU Scheduling II

1. Thread Scheduling - two basic levels
   (1) **Process local scheduling** (aka **process contention scope/unbound threads** available on the many-to-many and many-to-one models)
   (2) **System global scheduling** (aka **system contention scope/bound threads** available on the one-to-one model).



2. Multiple-Processor Scheduling
   • two configurations of multiple processors: **Asymmetric** (**master/slave-AMP**), and **Symmetric** (**SMP**)



• **Processor affinity**: the ability to direct a specific task, or process, to use a specified core.
The idea behind: if the process is directed to always use the same core it is possible that the process will run more efficiently because of the cache re-use.
➢ **Soft affinity** – OSs try to keep a process running on the same processor but not guaranteeing it will do so.
➢ **Hard affinity** - allows a process to specify a subset of processors on which it may run

• OS periodically checks and validates the numbers of process in ready queue. --> **Load Balancing**
-- If a processor is overloaded - to balance OS distributes the load by moving (or pushing) processes from overloaded to idle or less busy processors = **Push migration**
-- If scheduler finds there is no process in ready queue so it raids another processor's run queue and transfers a process onto its own queue so it will have something to

run (pulls a waiting task from a busy processor.) = **Pull migration**

- hyperthreading

SMP → multicore processors may complicate the scheduling problems.

↓

- processor spends a significant amount of time waiting for the data to become available (slowing or stopping of a process) = **MEMORY STALL.**

↓

The hardware solution = implement **multithreaded processor cores** in which hardware threads (additional cores -"*hyperthreading*") are assigned to each core.

- if one thread stalls while waiting for memory, the core can switch to another thread.

- two basic techniques for multithreading:

① **Coarse-grained multithreading** - switching between threads only when one thread blocks (long latency event such as a memory stall occurs).

② **Fine-grained multithreading** - instructions "scheduling" among threads obeys a Round Robin policy.

3. Real-Time CPU Scheduling

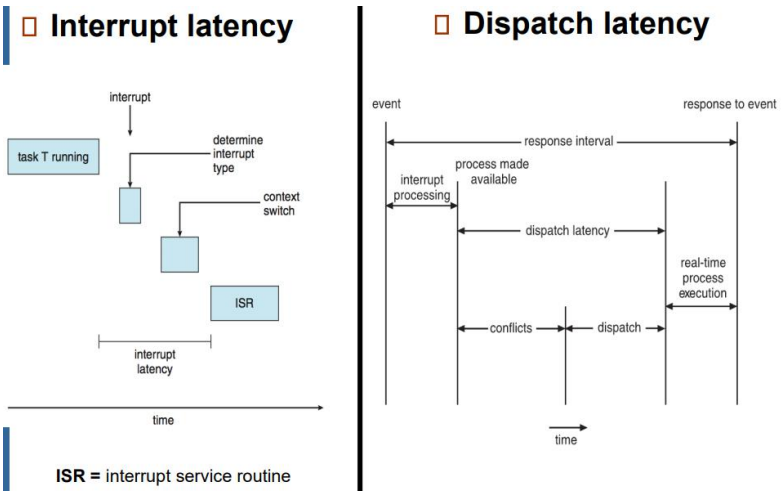
- **Hard real-time systems** – required to complete a critical task within a guaranteed amount of time.

Soft real-time computing – requires that critical processes receive priority over less fortunate ones.

- Two types of latencies affect performance

1) **Interrupt latency** – aka **interrupt response time** is the time that elapses from when an interrupt is generated to when the source of the interrupt is serviced.

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



- different types of real-time CPU scheduling

1) **Static scheduling** – is prepared before execution of the application begins. Process interactions, periodicities, resources constraints and deadlines are considered while forming the schedule

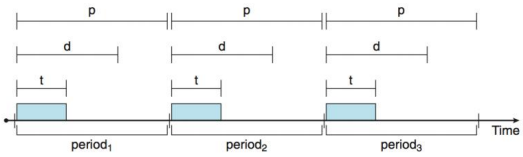
2) **Priority-based scheduling**. The real-time application is analysed and assign appropriate priorities to processes in it.

3) **Dynamic scheduling**. Scheduling is performed when a request to create a process is executed. Typically, a scheduling decision is performed when a process arrives.

- real-time CPU scheduling

1) Priority-Based Scheduling (preemptive scheduling)

- Process has new characteristics:
  - **Periodic** - requires CPU at constant intervals
    - processing time ***t***,
    - deadline ***d***,
    - period ***p***, $0 \le t \le d \le p$
  - **Rate** of a periodic task is $1/p$



2) **Rate-Monotonic Scheduling(RMS)**

- Schedules periodic tasks using a static priority policy with preemption.
- Static priorities are allocated during tasks / process creation.

  --The shorter the period = the higher the priority

  --The longer the period = the lower the priority
- The CPU utilization of a process P_i: t_i/p_i

  t_i = the processing time & p_i = the period

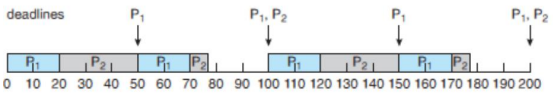
Two processes: P1 and P2.

- **P1 is assigned a higher priority than P2** (the period of P1 is shorter than that of P2).

P1 $p_1 = 50$, $t_1 = 20$ the CPU utilization of P1 = 20/50 = 0.4

P2 $p_2 = 100$, $t_2 = 35$. the CPU utilization of P2 = 35/100 = 0.35

total CPU utilization – 75%



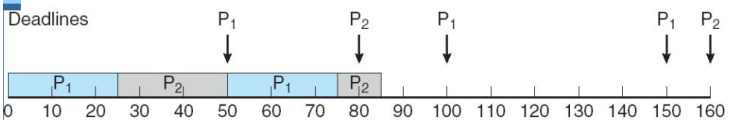
➢ Missed Deadlines with Rate Monotonic Scheduling

P1: $p_1 = 50$, $t_1 = 25$.

P2: $p_2 = 80$, $t_2 = 35$.

process P1 is higher priority.

The total CPU utilization (25/50)+(35/80) = 0.94



3) Earliest-Deadline-First Scheduling

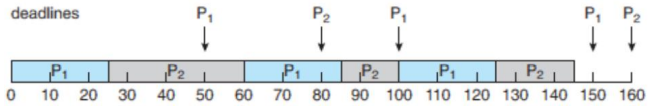
• dynamically assigns priorities according to deadline.

--the earlier the deadline = the higher the priority

--the later the deadline = the lower the priority.

P1: $p_1 = 50$, $t_1 = 25$.

P2: $p_2 = 80$, $t_2 = 35$.



4) Proportional Share Scheduling

• T shares are allocated among all processes in the system. An application receives N shares will receive N/T of the total processor time.

4. Algorithm Evaluation: How to select CPU-scheduling algorithm for an OS?

(1) Deterministic evaluation

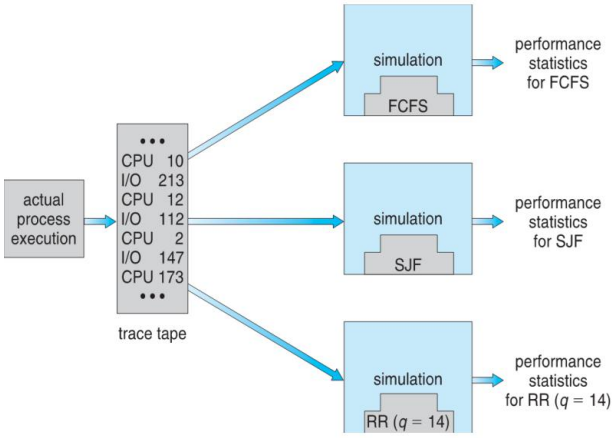
Takes a particular predetermined workload and defines the performance of each algorithm for that workload: What algorithm can provide the minimum average waiting time?

(2) Queueing Models

Little's law: processes leaving queue must equal processes arriving, thus: n = λ xW (n = average queue length, W = average waiting time in queue, λ = average arrival rate into queue)

Example: if on average 7 processes arrive per second, and normally 14 processes in queue, then average wait time per process = 2 seconds

(3) Simulations



Lecture6 Deadlocks

1. System Model

   Deadlock can be defined as the permanent blocking of a set of processes that compete for system resources

2. Deadlock Characterization

   MUTUAL EXCLUSION, HOLD AND WAIT, NO PREEMPTION, CIRCULAR WAIT

   - resource allocation graph with a deadlock:

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

3. Methods for Handling Deadlocks

   (1) Ensure that the system will never enter a deadlock state:

   ① Deadlock Prevention (miss one of deadlock conditions)

   --Mutual Exclusion: cannot be disallowed

   --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 allocated to it.

     --> Low resource utilization; starvation possible

   --No Preemption:

   - if a process holding certain resources is denied a further request, that process must release its original resources and, if necessary, request them again together with the additional resource.
   - if a process requests a resource that is currently held by another process, the OS may preempt the second process and require it to release its resources.

   --Circular Wait: can be prevented by defining a linear ordering of resource types.

   - if a process has been allocated resources of type R, then it may subsequently request only those resources of types following R in the ordering.

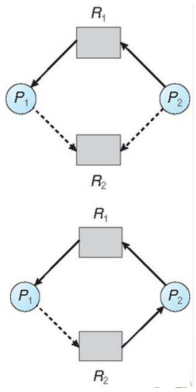
   ② Deadlock Avoidance(decide whether granting a resource is safe or not)

   - A **safe state** is one in which there is at least one sequence of resource allocations to processes that does not result in a deadlock-->no deadlock
     a) Do not start a process if its demands might lead to deadlock
     b) Do not grant an incremental resource request to a process if this allocation might lead to deadlock.
   - **resource-allocation graph algorithm**(each resource type has one instance)

Works only if **each resource type has one instance**

Algorithm:

- Add a claim edge $P_i \rightarrow R_j$ if P_i *can request resource* R_j *in the future.*
  - represented by a **dashed line** in graph
- A request Pi → Rj can be granted only if:
  - Adding an assignment edge Rj → Pi does not introduce cycles (since cycles imply unsafe state)



- **Banker's Algorithm**(each resource type has multiple instances)

  Consider each request to see whether granting it leads to a safe state, if yes, grant the resources to the process

  - Check that **Request ≤ Available** (that is, (1, 0, 2) ≤ (3, 3, 2) ⇒ true

	Allocation	Need	Available
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

  - Executing safety algorithm shows that sequence < P_1, P_3, P_4, P_0, P_2> satisfies safety requirement

(2) Deadlock Detection(grant resources to processes whenever possible)

- deadlock detection with <u>one resource of each type</u>: the system needs to maintain the **wait-for graph** and periodically invoke an algorithm that searches for a cycle in the graph (circular wait condition).
- deadlock detection with <u>several instances of a resource type</u>: invoke detection algorithm(similar to bankers' algorithm)-->decide when & how often to invoke the detection algorithm

(3) Recovery from Deadlock

- System recover from the deadlock automatically, are two options for breaking a deadlock:
  a. To <u>abort</u> one or more processes to break the circular wait: abort one/all processes at a time
  b. To <u>preempt</u> some resources from one or more of the deadlocked processes:
     i. <u>Selecting a victim</u>: need to minimize cost
     ii. <u>Rollback</u>: return to some safe state
     iii. <u>Starvation</u>: avoid same process may always be picked as victim

Lecture8 Memory Management: Main Memory

1. Background

• **relocation**: the process of moving a program from one area of memory to another area. The base register is now termed a **relocation register**, physical address = logical address + relocation register.

4. Logical organization of memory: Allocation

(1) contiguous memory allocation - Partitioning

• Fixed/Static Partitioning: fixed size blocks -->Internal fragmentation

• Variable/Dynamic Partitioning: variable size segments -->External fragmentation <--Compaction, Non-contiguous memory allocation: Segmentation & Paging

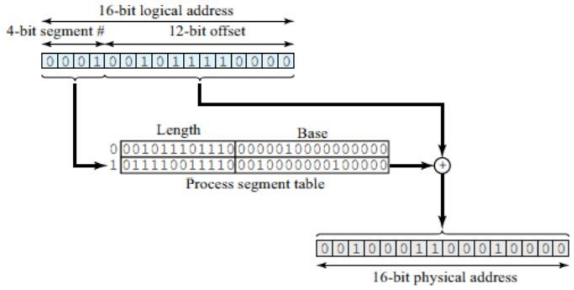
--Placement Algorithm: First-Fit, Best-fit, Worst-fit

(2) Non-contiguous memory allocation: Segmentation & Paging

① Segmentation:

• Logical address: <segment-number, offset> = <s, d>

• Segment table: (base+limit) maps two-dimensional logical address to one-dimensional physical address.



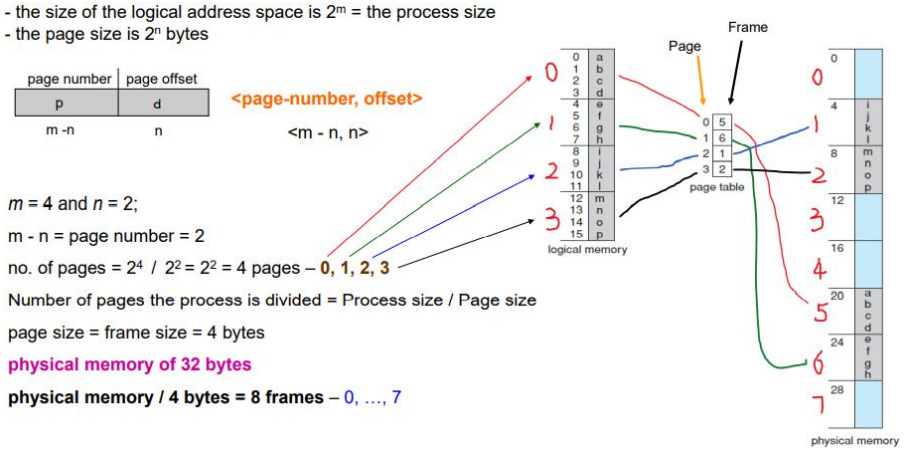
② Paging:

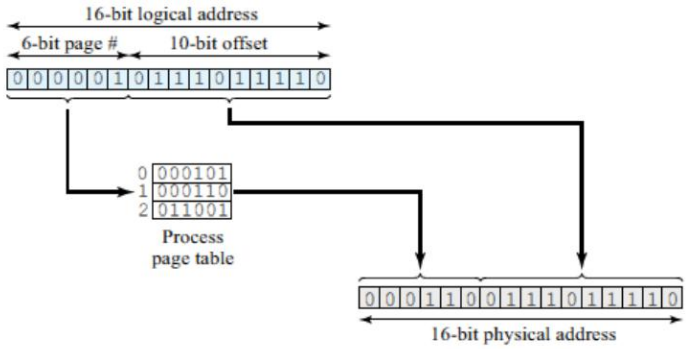
• Memory is divided into equal-size partitions called frames.

• Logical address (space) is divided into blocks of same size as frames called pages of a process

• logical address: <page-number, offset> = <p,d>

• page table:





- free frames allocation
- **TLB**
    a. Total memory access time = Time to access page table(PTBR: Page-table base register) + Time to access memory location
    b. Translation Look-aside Buffers(TLBs): a special fast-lookup hardware cache

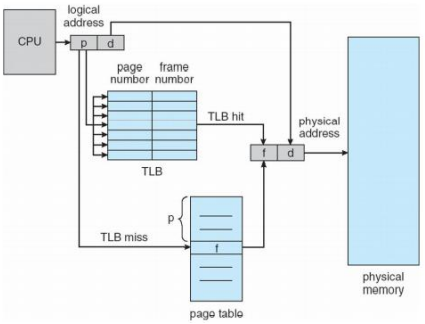
The percentage of times that the page number of interest is found in the TLB is called the **hit ratio**.

- **TLB lookup time (E)**,
- **memory access time (M)** and
- **hit ratio (A)**

Estimation of the impact of the TLB on the execution speed of the computer - **the effective memory-access time (EAT)**

$$EAT = A(E + M) + (1 - A)(E + 2 \times M)$$

1 - A = TLB **miss ratio**



- Valid (v) or Invalid (i) Bit In A Page Table

    Valid-invalid bit attached to each entry in the page table: “valid” indicates that the associated page is in the process logical address space, and is thus a legal page/ indicates page is in the main memory

- Shared pages

    --Efficient communication: Processes communicate by write to shared pages

    --Memory efficiency: One copy of read-only code/data shared among processes

- large page table problem

    --Hierarchical Paging: break up virtual address space into multiple page tables at different levels. (Two-Level Paging Example)

    --Hashed Page Tables: the virtual page number is hashed into a page table. This page table contains a chain of elements hashing to the same location

    --Inverted Page Tables: stores a process Id of each process to identify its address space uniquely.

Lecture9 Memory Management: Main Memory

1. Background
   - **resident set**: the components of a process that are present in the memory

2. Demand paging: loading only a part of the program into memory for processing.
   - **Lazy swapper**: never swaps a page into memory unless page will be needed.Swapper that deals with pages is a pager.
   - How to recognize whether a page is present in the memory? -->Set to "valid"/"invalid"
   - What happens if the process tries to access a page that was not brought into memory?-->cause a **trap** to the OS so that a **page fault** can be noticed
   - performance of demand paging-->The Effective Access Time (EAT) EAT = (1 – p) x Memory Access + p x page fault time

3. **Copy-on-Write(COW)**: technique used in sharing the virtual memory of operating system processes, (e.g. the implementation of the fork system call).
   - The parent and child process to share the same pages of the memory initially
   - If any process either parent or child modifies the shared page, only then the page is copied.

4. Page replacement
   - find location of desired page on disk --> no free frame, use a **page replacement algorithm** to select a victim frame and swap it out.(Check the **modify(dirty) bit**: if set(has been modified), write the page into disk, otherwise, discard the page.)-->change to invalid -->swap desired page in
   (1) First-In First-Out (FIFO) Algorithm
   (2) **Optimal Algorithm**: Replace page that will not be used for longest period of time
   (3) Least Recently Used (LRU) Algorithm: throw out the page that has been unused for the longest time-->**counter implementation**(a time-of-use field or a counter)/**stack implementation**
   (4) Enhanced Second-Chance (Clock) Algorithm-->iterator scan: if page's RB = 1, set to 0 & skip; else if RB = 0, remove (**RB=Reference bit**: is set by the hardware and will say whether the page has been referred in the last clock cycle or not.)
   (5) Counting-Based Page Algorithms: Least Frequently Used (LFU) Algorithm & Most Frequently Used (MFU) Algorithm
   - **Reference string** : is the sequence of pages being referenced.

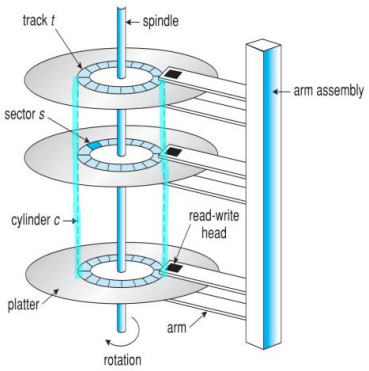
5. Frames Allocation (How many frames to allocate to each process?)
   - **thrashing**： high paging activity. A process is thrashing if it is spending more time paging than executing.
   (1) Fixed allocation
      ① Equal allocation
      ② Proportional allocation
   (2) Priority allocation: page fault-->select a frame with lower priority number

Lecture10 Mass-Storage Systems

1. Overview:

• types of secondary storage: sequential access devices , direct access devices(Less capacity but much faster than HDDs)

• moving-head disk mechanism



- Group of tracks is called a **cylinder.**
- Aluminum **platters** with magnetic coating.
- A stack of **16 platters** is about the maximum one will find in modern drives.
- A track is logically divided into **sectors**.
- The sectors are the smallest unit of data that a disk drive will transfer.
- **Disk address** can be specified by the **cylinder, head** and **sector numbers**, or **CHS addressing**.

A disk with **C** cylinders, **H** heads, and **S** sectors per track has **C x H x S** sectors in all

• disk speed

Transfer rate = data rate between disk drive and computer

Seek time = the time taken by the disk head to move from one cylinder to another

Rotational latency = the time to rotate to the sector

Random access time/Positioning time = seek time + rotational latency

Disk Access Time = the time to perform any operation on the disk.

seek time + rotational latency + transfer time.

2. Disk structure

• Disk is addressed as a one-dimension array of logical sectors.

--Logical sector 0: the first sector of the first (outermost) track of the first surface

• **Disk controller** maps logical sector to physical sector identified by track #, surface # and sector # .

3. Disk attachment 磁盘连接

(1) via IO ports on small systems: Host-attached storage(IDE, ATA)

(2) via a remote host in a distributed file systems: Storage Area Network, Network-Attached Storage(TCP/IP, UDP/IP, ISCSI)

4. Disk scheduling: performed by O.S. and disk itself

(1) First-Come First-Served(FCFS) --> Fairest of them all

(2) Shortest Seek Time First(SSTF) --> High utilization, small queues

(3) SCAN(Elevator): from one end to the other --> Better service distribution

(4) C-SCAN --> Lower service variability
(5) LOOK: from the first request to the last
(6) C-LOOK

5. Disk management
   (1) Low-level formatting, or physical formatting — create sectors on a blank platter(Each sector can hold header information, plus data, plus **error correction code** (**ECC**))
   (2) Partition organize disk in one or more groups of cylinders
   (3) Logical formatting write file system data structures
   (4) **Boot block** initializes system(The bootstrap is stored in ROM. **Bootstrap loader** program stored in boot blocks of boot partition)

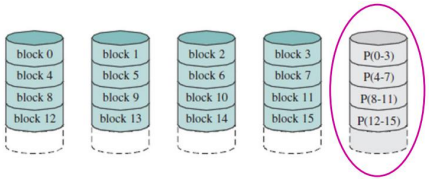
6. Swap-Space Management
   - **Swap-space**: Virtual memory uses disk space as an extension of main memory
   - Configure Swap-space:
     - ➢ on a swap file in a file system --> changing the size of a swap file is easier.
     - ➢ on a separate swap partition --> swap partition is faster but difficult to set it up (how much swap space your system requires?)
       -->Solution: start with a swap file and create a swap partition when it knows what the system requires.
   - Swap-space management
     -->Kernel uses swap maps to track swap-space use

7. **RAID** Structure (Redundant Arrays of Independent Disks)
   - a system of data storage that uses multiple hard disk drives to store data.
   - a set of physical drives viewed by the operating system as a single logical drive
   - levels: different storage methods
   - RAID controller: controlling a RAID array. Maybe hardware- or software-based.
   - **Mirroring**: writing data to two or more hard drive disks (HDDs) at the same time-->preserves the data from the failed disk

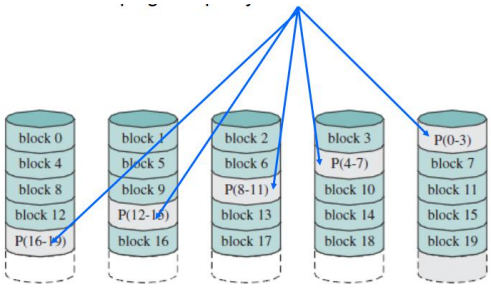
   Striping: breaks data into "chunks" that are written in succession to different disks-->high data-transfer rates
   - RAID is secure & fast. Is not a backup

   (1) RAID Level 0：treats multiple disks as a single partition. Files are Striped across disks, no redundant info. --> high read throughput, data loss
   (2) RAID Level 1(**mirrored configuration**): disk mirroring, use striping
   (3) RAID Level 2(**memory-style error-correcting-code organization**): an error-correcting code is calculated across corresponding bits on each data disk, and the bits of the code are stored in the corresponding bit positions on multiple parity disks. uses very small strips (often the size of a word or a byte)
   (4) RAID Level 3(**bit-interleaved parity organization**): a modification of Level 2 and requires only a single redundant disk. single **parity bit**(Suppose, a strip X = {1010}, the parity bit is 0 as there are even number of 1s) can be used for error correction / detection for each strip, and it is stored in the dedicated parity disk

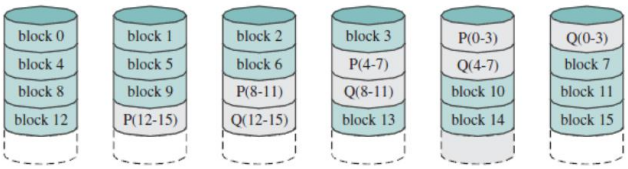
(5) RAID Level 4(**block-interleaved parity organization**): uses large-sized strips, and the data is striped as fixed-sized blocks. provides block-level striping (like Level 0) and stores a parity block on a dedicated disk.



(6) RAID Level 5(**block-interleaved distributed organization**): a modification of Level 4:the parity bits are not stored in a single devoted, - distributes striping and parity at a block level.



(7) RAID Level 6(**P + Q redundancy scheme**):independent data disks with double parity(provide two different parity calculations: one is the same as used in Level4&5, the other is an independent data-check algorithm)



(8) RAID Level 0+1(a mirror of strips)

(9) RAID Level 1+0(a strip of mirrors) -->advantage: in case of failure of a single disk, the mirror copy of the whole disk is available -->disadvantage: costly

Lecture11 Storage Manage: File-System

1. File concept
   - File types
   - File Attributes: name, identifier, type, location, size, protection, time, date and user identification
   - File Operations: create, write, read, save, reposition within file, delete, truncate, open(open file table), close

     (Reposition within file : the current position pointer used to read and write in the file can be repositioned to any desired position. The general system call for this operation is seek in some OSs. Therefore, it is also known as **seek operation**.)

     (Truncate: to erase the contents of a file but keep its attributes.)
   - logical categories: shareable vs. unshareable files, variable vs. Static files

2. Access Methods: Sequential Access, Direct Access, Indexed Access Acheme

3. Directory and Disk Structure:
   (1) Storage structure:
      ① A disk can be used in its entirety for a file system
      ② A physical disk can be broken up into multiple partitions, slices, or mini-disks, each can have its own filesystem.
   (2) Operations Performed on Directory: search,create, delete, list, rename, traverse
   (3) Schemes of logical structure of a directory: single-level directory, two-level directory(master directory+user directory), hierarchical/tree-structured directories, acyclic-graph directories(must allow sharing)

4. File-system mounting

   This attaching of the device is known as **mounting**, and the directory where the device is attached, is known as a mount point

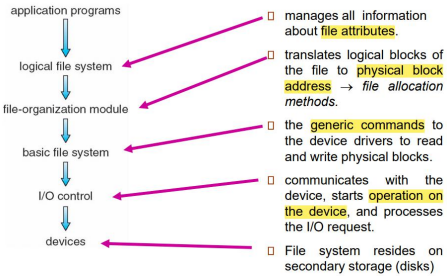
5. File Sharing
   - Multiple Users: file owner(or user) and group
   - Remote File Systems: FTP, distributed file systems, world wide web --failure modes: stateless protocols

     -->Client-server model: **Network File System NFS**(UNIX client-server file sharing protocol), **Common Internet File System-CIFS**(standard windows protocol)

     -->Distributed Information Systems: implement unified access to information needed for remote computing (LDAP, DNS, NIS, Active Directory)

6. Protection: identity dependent access: associate with each file and directory an **access-control list(ACL):** username(owner/group/public access) + access type(read,write,execute:RWX)

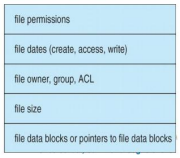
7. File-System Structure



8. File System Implementation

- on-disk for data storage: **File Control Block**(**FCB**)

- **Boot control block –** loads the OS in that partition.
- **File Control Block** (FCB) is required to track the information such as which data blocks comprise a file, and the file's size, owner, access rights, date of creation, access or modification, and so on.
- **Volume control block** stores the key information about the file system on every partition (or volume) of the disk
- **Free list** is used for tracking unallocated blocks.
- **Files and directories**



A Typical File Control Block

- in-memory for data access: **system-wide open-file table**(**SOFT**), **per-process open-file table**(**POFT**)

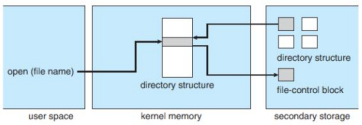
- A **mount table** that stores file system mounts, mount points, file system types.

- A **system-wide open-file table SOFT** maintains the information about the open files in the system. It keeps a copy of the FCB for each open file.

- A **per-process open-file table POFT** maintains the detail of every file opened by a process and an entry in the POFT points to a SOFT.

- **Buffers** is a temporary storage area in the memory for assisting in the reading/writing of information from/to disk.

Open a file (steps)



- It searches the SOFT to find out whether this file is already in use by another process or not.
- If **YES**, then an entry in the POFT is created pointing to the SOFT.
- If the file is **NOT FOUND** in the SOFT, then to locate the FCB of the desired file, the FCB of the directory/root is read in the memory.
- The FCBs of all the directories or sub-directories are searched until the desired FCB is found.
- It is copied to the SOFT in the memory. An entry in POFT pointing to the entry in SOFT is also created.
- The file system returns a pointer to the appropriate entry in the POFT. This pointer is known as a **file descriptor** or **file handle**.

9. Allocation Methods: how disk blocks are allocated to files

(1) Contiguous --> simple, external fragmentation(-->compaction off-line(downtime) or on-line)

(2) Linked: **file allocation table**(**FAT**) --> only efficient for sequential access

(3) Indexed: **index block**(big index block problem --> linked scheme, Multi-Level Index,

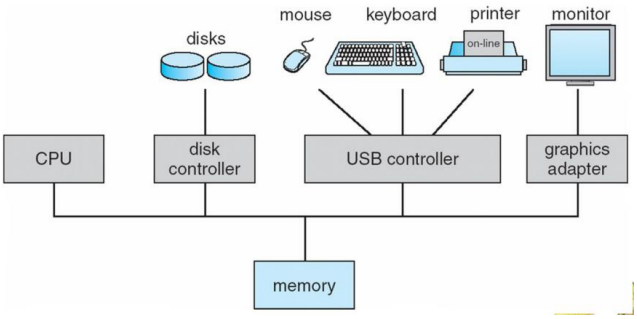
Combined Scheme)

10. Free-Space Management

- **Bit Vector** - each bit represents a disk block, set to **1 if free** or **0 if allocated**.
- **Linked List** - **link together all the free disk blocks**, keeping a pointer to the first free block **.**
- **Grouping** - stores the **addresses of *n* free blocks in the first free block.**
- **Counting** - **the number of contiguous free blocks**.
- **Space Maps** - free-space list is implemented as a **bit map**, bit maps must be modified both when blocks are allocated and when they are freed.

Lecture12 Mass-Storage Systems

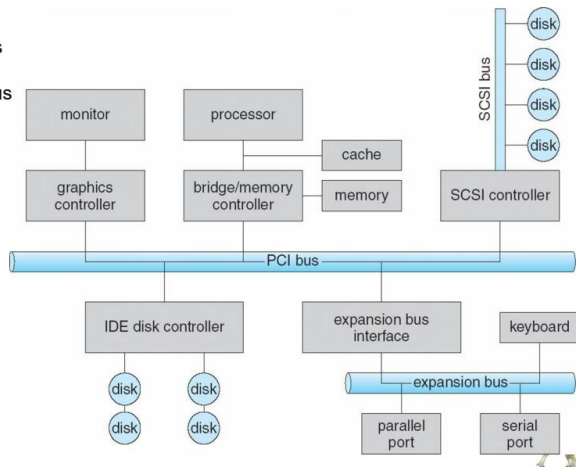
1. Overview
   - Ports, busses, device controllers connect to various devices
     Device drivers encapsulate device details, Present uniform device-access interface to I/O subsystem
   - I/O devices



   Classified:
   1) Human-readable & machine readable
   2) Transfer of data: character-oriented device, block-oriented device
   3) Types pf access: sequential device(tape driver), random access device(disk)
   4) Network device: to send or receive data on a network

2. I/O Hardware
   - Incredible variety of I/O devices: Storage, Transmission, Human-interface
   - common concepts
     --port: connection point for device
     --bus: daisy chain or shared direct access(expansion bus connects relatively slow devices)
     --controller(host adapter): electronics that operate port, bus, device(sometimes integrated, sometimes separate circuit board(host adapter))

- PCI bus
- expansion bus
- SCSI bus
- daisy-chain bus



• each controller has registers that are used for communicating with the CPU, many devices have a data buffer that the operating system can read and write.

-->four types of registers

a. **Data-in/out register**: holding data until the CPU, or the output device, is ready to accept it
b. **Status register**: providing information to the CPU on the status of the I/O device
c. **Control register**: concerned with transferring I/O commands between the CPU and an I/O device

• How the CPU communicates with the control registers and the device data buffers?

- **Direct I/O instructions**
  - each register is assigned an **I/O port number** (8- or 16-bit integer)
  - using I/O instructions (IN / OUT) the CPU can read/write the contents in control register PORT
- **Memory-mapped I/O**
  - **map all the registers into the memory space**
  - communications occur by reading and writing directly to/from those memory areas.

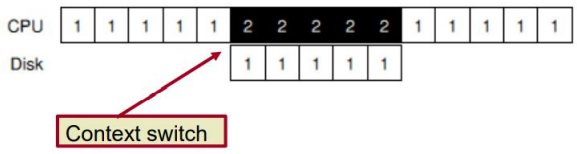
• **I/O communication techniques**: techniques by which an I/O operation can be performed on a device(these techniques are used to have a mode of communication between the user request and the device)

① **Polling**: CPU executes a busy-wait loop - periodically tests the channel status bit

② **Interrupt-driven I/O**

Through an **interrupt mechanism** (see *Interrupt-driven I/O Cycle*).

- when the operation is complete, the device controller generates an interrupt to the processor.
- the processor checks for the interrupt after every instruction cycle.
- after detecting an interrupt, the processor will stop what it was doing by saving the state of the current process and resumes the previous process, by executing the appropriate **interrupt service routine**.
- **Interrupt handler** receives interrupts the processor then performs the data transfer for the I/O operation.



--two interrupt lines:

1) Non-maskable - for critical error conditions
2) Maskable - used by device controllers to request

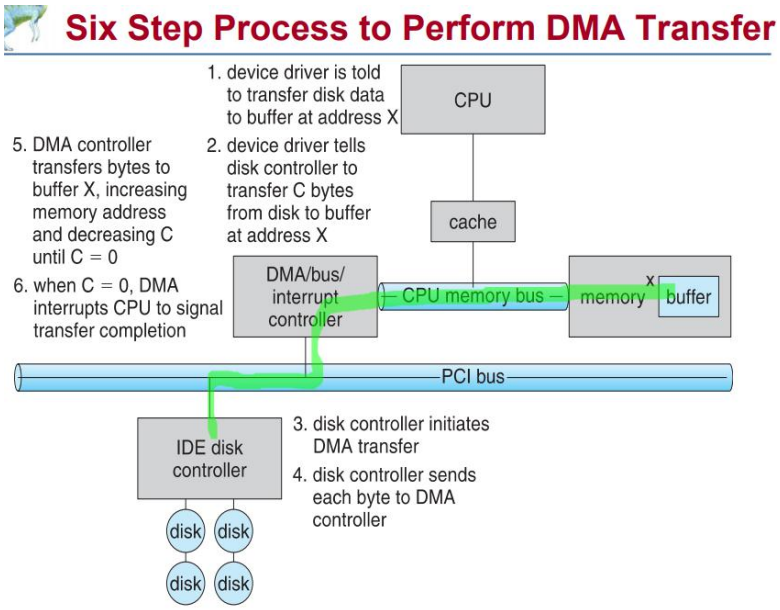
--basic protocol to interact with the device: wait for drive to be ready, write parameters to control registers, start the i/o; data transfer; handle interrupts, error handling

③ **Direct Memory access(DMA)**: bypass CPU to transfer data directly between I/O device and memory

- Require **DMA controller**. (Version that is aware of virtual addresses

can be even more efficient - **Direct Virtual Memory Access DVMA**)

• the processor passes the following information to the DMA controller: type of request(read/write) + address of the I/O device + start address of the memory along with total words number



2. Application I/O Interface
   - Devices-driver layer hides differences among i/o controllers from kernel
   - characteristics of i/o devices

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read–write	CD-ROM graphics controller disk

--block and character devices

- **Block devices** are accessed a block at a time
  - include disk drives
  - Commands include **read, write, seek**
    - *Raw I/O* (accessing blocks on a hard drive directly),
    - *Direct I/O* (uses the normal filesystem access)
    - Memory-mapped file I/O.
- **Character devices** are accessed one byte at a time
  - include keyboards, mice, serial ports
  - Commands include `get(), put()`
  - supported by higher-level library routines.

--network devices

- Varying enough from block and character to have own interface
- Linux, Unix, Windows and many others include **socket interface**
  - *socket* acts like a cable or pipeline connecting two networked entities.
- Approaches vary widely (pipes, FIFOs, streams, queues, mailboxes)

--Clocks and Timers

- Three types of time services are commonly needed in modern systems:
  - Get the current time of day.
  - Get the elapsed time since a previous event.
  - Set a timer to trigger event X at time T.
- A **programmable interrupt timer**, PIT can be used to trigger operations and to measure elapsed time.
- **ioctl()** (on UNIX) covers odd aspects of I/O such as clocks and timers

--**Blocking and Non-blocking I/O**

- Blocking - process suspended (move it in the waiting queue) until I/O completed
- Nonblocking: the i/o request returns immediately-->allow the process to continue on other tasks and then the process is notified when the i/o operation has completed and the data is available for use. (Asynchronous I/O = variation of the non-blocking I/O)

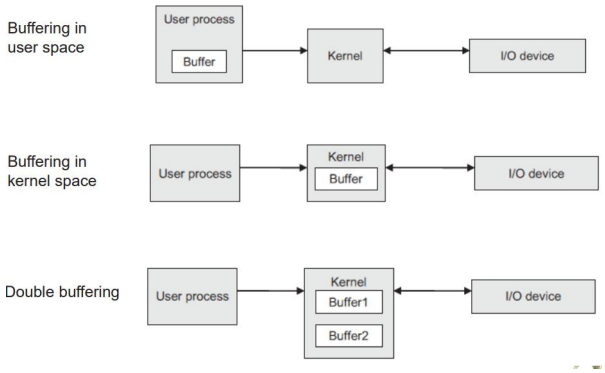
3. Kernel I/O Subsystem

-->kernel keep state info for i/o components, including open file tables, network connections, character device state,…

- Scheduling: i/o request ordering
- **buffering**: store data in memory while transferring between devices

-->operation on the device can be performed with its own speed

1) Single buffering: kernel and user buffers, varying sizes
2) Double buffering(two copies of the data): one set of data to be used while another is collected



- **Caching**: faster device holding copy of data

  --keep a copy of data in a faster-access location

  --buffering and caching use often the same storage space

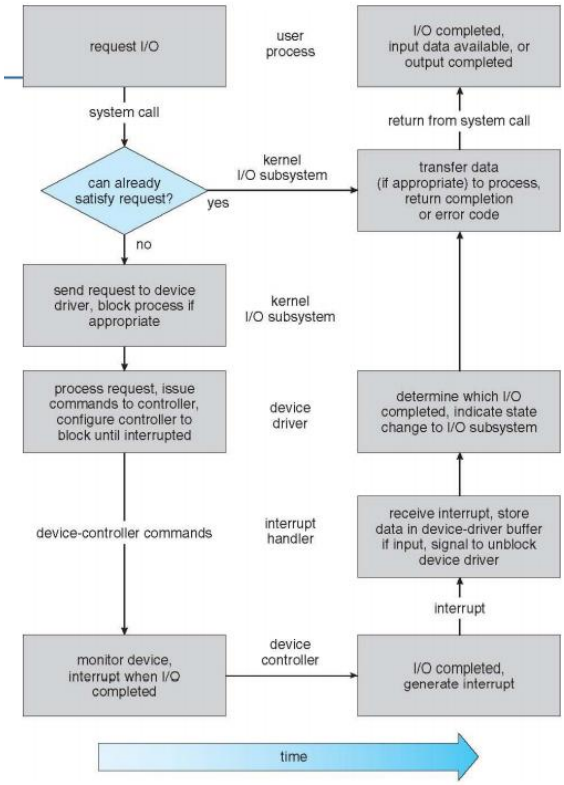
- **Spooling**: Simultaneous Peripheral Operations On-Line buffers data for devices that cannot support interleaved data streams.
- **device reservation**: provide exclusive access to a device

  --system calls for allocation and de-allocation

  --watch out for deadlock
- **Error Handling**: use protected memory can guard errors

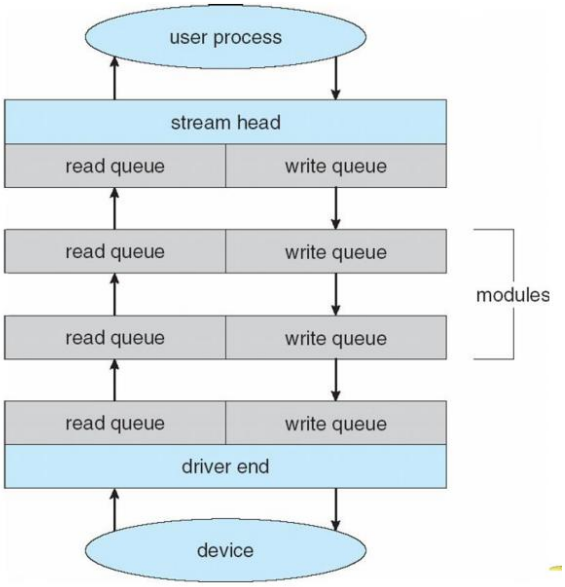
  --i/o requests usually return an error bit(or more)indicating the problem

  --UNIX systems also set the global variable errno
- **i/o protection**

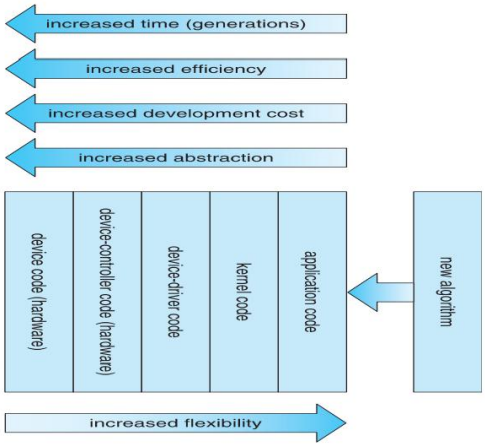


4. Streams: a full-duplex communication channel between a user-level process and a device in Unix System V and beyond

- a stream consists of:
  - --the user process interacts with the **stream head**
  - --the device driver interacts with the **device end**
  - --zero or more **stream modules** between(each module contains a read queue and a write queue)
- message passing is used to communicate between queues: flow control can be optionally supported. Without flow control, data is passed along as soon as it is ready.



5. Improve Performance



Lecture13 Protection & Security in OS

1. Goals of Protection

2. Principles of Protection:

   •**principle of least privilege**(programs, users and systems be given just enough privileges to perform tasks): can be static or dynamic(**domain switching, privilege escalation**)

3. Domain of Protection

   • domain structure:

   ① Protection domain: the resources that a process may access

   ② Access right: the ability to execute an operation on an object

   ③ Domain: a set of <objects,{access right set}> pairs

   --> Access Matrix: Access(i,j)-->set of operations that a process executing in $Domain_j$ can invoke on $object_j$

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

   -->implementation: global table, access lists for objects, capability lists for domains, a lock-key mechanism(Each resource has a list of unique bit patterns, termed locks. ▪ Each domain has its own list of unique bit patterns, termed keys. ▪ Access is granted if one of the domain's keys fits one of the resource's locks.)

   • access control:

   Role-Based Access Control(RBAC): assign privileges to users, programs, or roles as appropriate

   --privileges refers to the right to call certain system calls, or to use certain parameters with those calls

   --privileges and programs can also be assigned to roles

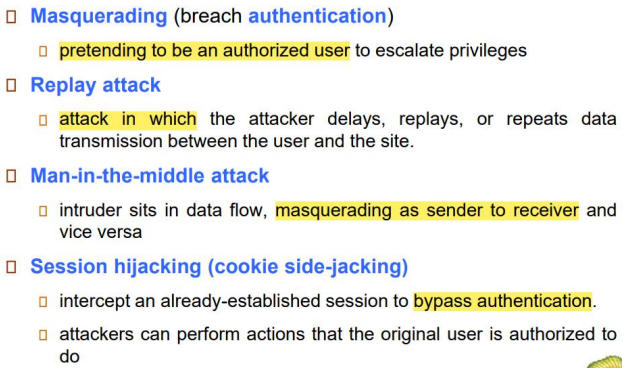
   --Role-based access control (RBAC) is a security feature for controlling user access to tasks that would normally be restricted to the root user.

   • options to revocate access rights: immediate & delayed, selective & general, partial & total, temporary & permanent

4. The Security Problem

• security violation categories: Breach of confidentiality-->Unauthorized reading of data, Breach of integrity-->Unauthorized modification of data ,Breach of availability-->Unauthorized destruction of data, Theft of service-->Unauthorized use of resources,Denial of service (DOS)-->Prevention of legitimate use

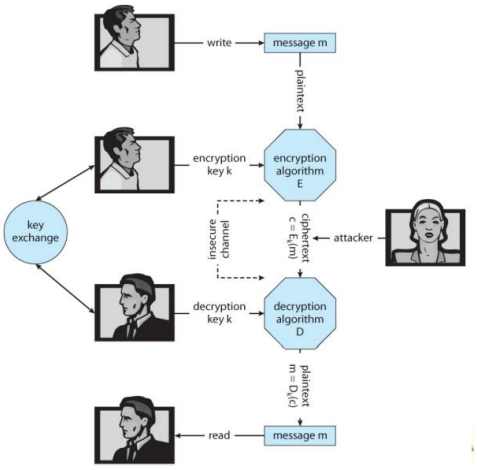
• security violation methods:



• security measure levels: physicals, human(avoid social engineering, phishing, password cracking,dumpster diving), operating system, network

5. Threats
   (1) Program Threats
      ① **Trojan Horse** - exploits mechanisms for allowing programs written by users to be executed by other users (Spyware, pop-up browser windows, covert channels)
      ② **Trap Door** - is when a designer or a programmer (or hacker) deliberately inserts a security hole that they can use later to access the system.
      ③ **Logic Bomb** - is a piece of code intentionally inserted into a software system that will set off a malicious function when specified conditions are met.
      ④ **Stack and Buffer Overflow** exploits a bug in a program (overflow either the stack or heap buffers)
      ⑤ **Viruses** - fragment of code embedded in an otherwise legitimate program, designed to replicate itself (by infecting other programs), and (eventually) wreaking havoc.
   (2) System and Network Threads
      ① **Worm** - is a process that uses the fork / spawn process to make copies of itself in order to wreak havoc on a system. Worms consume system resources, often blocking out other, legitimate processes.
      ② **Port Scanning** is technically not an attack, but rather a search for vulnerabilities to attack.
      ③ **Denial of Service (DOS)** attacks do not attempt to actually access or damage systems, but merely to clog them up so badly that they cannot be used for any useful work. DOS attacks can also involve social engineering.

6. Cryptography as a Security Tool



(1) Symmetric Encryption: same key used to encrypt and decrypt (Data Encryption Standard(**DES**), Triple-DES)

(2) Asymmetric Encryption: public-key encryption based on each user having two keys: public key--published key used to encrypt data, private key--key known only to individual user used to decrypt data(most common: **RSA**)

(3) Authentication: verify the identity of the entity who transmitted a message.

• the signing function: produces an authenticator: a value to be used to authenticate a message.

• a verification: produces a value of "true" if the authenticator was created from the message, and "false" otherwise

-->two main varieties of authentication algorithms:

a) **Message Authentication Code (MAC)**: uses symmetric encryption.

-- a cryptographic checksum is generated from the message using a secret key.

b) **Digital-signature algorithm** - uses asymmetric encryption.

-- A person can encrypt signature related data with the use of a private key

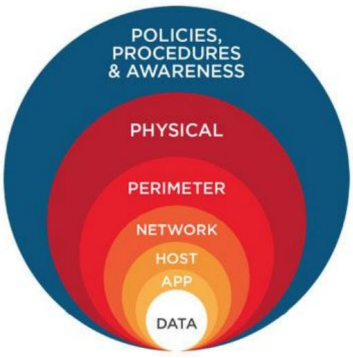
-- One can give the public key to anyone who needs verification of the signer's signature

(4) Key Distribution

- Key distribution with **symmetric cryptography** is a **major problem**
  - One option is to send them *out-of-band*, say via paper or a confidential conversation
- **Asymmetric keys** solves some of these problems, because the public key can be freely transmitted through any channel, and the private key doesn't need to be transmitted anywhere.
  - the public keys are not confidential, so this **key-ring** can be easily stored and managed (**key-ring** is simply a file with keys in it.).
  - even asymmetric key distribution needs care – *man-in-the-middle attack*

- is the public key safe?-->Digital certifications
  - **Proof of who or what owns a public key**
  - **Public key** digitally signed a trusted party
  - **Trusted party** receives proof of identification from entity and certifies that public key belongs to entity
  - **Certificate authority** are trusted party – their public keys included with web browser distributions
    - They vouch for other authorities via digitally signing their keys

7. User Authentication
   - passwords: password vulnerabilities, securing passwords, one-time passwords
   - Biometrics: involve a physical characteristic of the user
8. Implementing Security Defenses
   - **Defense in depth** is most common security theory – **multiple layers of security**



- improve security: security policy, vulnerability assessment, intrusion detection 入侵检测, virus protection, Auditing, Accounting, and Logging 审计，记账和日志记录

Intrusion Detection

- Intrusion detection attempts to detect attacks, both successful and unsuccessful attempts.
  - *Signature-based* scans network packets, system files, etc. looking for recognizable characteristics of known attacks
  - *Anomaly detection* looks for "unusual" patterns of traffic or operation
- **Intrusion Detection Systems**, IDSs, raise the alarm when they detect an intrusion.
- **Intrusion Detection and Prevention Systems**, IDPs, act as filtering routers, shutting down suspicious traffic when it is detected.

Lecture14 Virtual Machines & Distributed Systems

1. Virtual Machines

• **Virtualization** is technology that allows to create multiple simulated environments or dedicated resources from a single, physical hardware system.

•Software called a **hypervisor** connects directly to that hardware and allows to split a system into separate, distinct, and secure environments known as **virtual machines (VMs**).

• **Virtual machine manager (VMM)** or **hypervisor** – creates and runs virtual machines by providing interface that is identical to the host (except in the case of paravirtualization)

Virtual machine implementations involve several components

-- Host – the physical hardware equipped with a hypervisor.

-- Guest – an operating system

--Single physical machine can run multiple operating systems concurrently, each in its own virtual machine

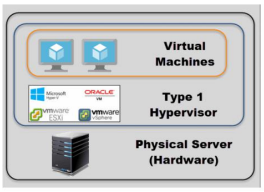
• the hypervisor provides a layer between the hardware (the physical host machine) and the Virtual Machines (guest machines)

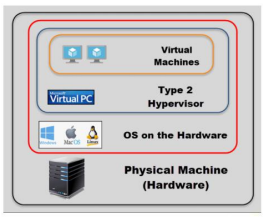
2. Types of Virtual Machines

Types of virtual machine manager VMMs:

- **Type 0 hypervisors** - *Hardware-based solutions* that provide support for virtual machine creation and management via firmware.

- **No need an embedded host OS** to support virtualization, runs in an "**Un-Hosted" environment**.
  - IBM LPARs and Oracle LDOMs are examples
- **Type 1 hypervisors** - *Operating-system-like software* built, **is a layer of software run directly on the system hardware.**
  - Including VMware ESX, Joyent SmartOS, and Citrix XenServer
  - Including Microsoft Windows Server with HyperV and RedHat Linux with KVM
- **Type 2 hypervisors** - **allows users to run multiple operating systems simultaneously on a single platform**
  - Including VMware Workstation and Fusion, Parallels Desktop, and Oracle VirtualBox
- Other variations include:
  - **Paravirtualization** - technique in which the guest operating system is modified to work in cooperation with the VMM to optimize performance
  - **Programming-environment virtualization** - VMMs do not virtualize real hardware but create an optimized virtual system
    - used by Oracle Java and Microsoft.Net
  - **Emulators** – allow **applications written for one hardware environment to run on a very different hardware environment**, such as a different type of CPU
  - **Application containment** - provides virtualization-like features by **segregating applications from the operating system**, making them more secure, manageable
    - Including Oracle Solaris Zones, BSD Jails, and IBM AIX WPARs





3. Benefits

- the ability to share the same hardware yet run several different execution different operating systems concurrently.

- Host system protected from VMs, VMs protected from each other
  - against virus - less likely to spread
  - each virtual machine is almost completely isolated from all other virtual machines

Disadvantage of isolation is that it can prevent sharing of resources.

- a perfect for operating-system research and development.

- virtualized workstation allows for rapid porting and testing of programs in varying environments.

- Consolidation involves taking two or more separate systems and running them in virtual machines on one system.

- can improve resource utilization and resource management.

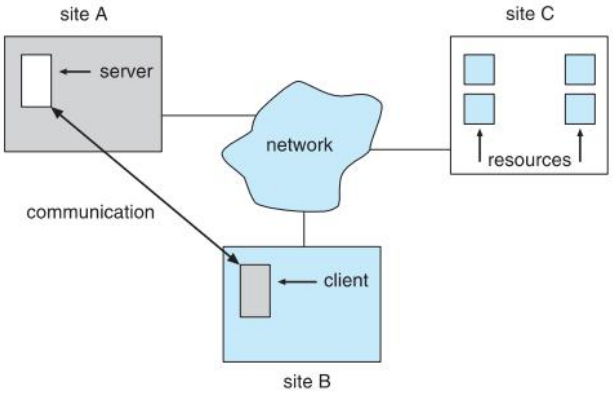
- Live migration – move a running VM from one host to another.

4. Distributes Systems

Distributed system is a loosely-coupled architecture, wherein processors are inter-connected by a communication network.

--Processors variously called nodes, computers, machines, hosts

--The processors and their respective resources for a specific processor in a distributed system are remote, while its own resources are considered as local.



5. Reasons for Distributed Systems

- Resource sharing
  - » Sharing and printing files at remote sites
  - » Processing information in a distributed database
  - » Using remote specialized hardware devices
- Computation speedup – load sharing or job migration (are distributed and run concurrently on various nodes on the system)
- Reliability – detect and recover from site failure, function transfer, reintegrate failed site; may utilize an alternative path in the network, in case of any failure.
- Communication – exchange information at geographically-distant nodes
- Economy and Incremental growth - a number of cheap processors together

provide a highly cost-effective solution for a computation intensive application. The DS may be increased with the introduction of any new hardware or software resources.

6. Types of Network-oriented Operating Systems
    - Network Operating Systems
        - ➢ Remote logging into the appropriate remote machine (telnet, ssh)
        - ➢ Remote File Transfer - transferring data from remote machines to local machines, via the File Transfer Protocol (FTP) mechanism
        - ➢ Users must establish a session
    - Distributed Operating Systems
        - ➢ Data Migration – transfer data by transferring entire file, or transferring only those portions of the file necessary for the immediate task
        - ➢ Computation Migration – transfer the computation, rather than the data, across the system
        - ➢ Process Migration – execute an entire process, or parts of it, at different sites