Grading

- **Midterm: 25%**
  - Theory Part 60% (1h 30m)
  - Programming Part 40% (1h)

- **Theory Part (Closed Books):**
  - Similar to your assignments
  - Questions will cover all topics

- **Programming Part (Open Books):**
  - Debug given code & Add functionality (Select C or Java)
  - Explain your changes
  - Emphasis on logic not on syntax
Midterm Topics

- Introduction, Threads and Processes
- Inter-process Communication, Synchronization
- CPU Scheduling
- Deadlocks
Parallel Systems

- Multiprocessor systems with more than one CPU in close communication.

- *Tightly coupled system* – processors share memory and a clock; communication usually takes place through the shared memory.

- Advantages of parallel system
  - Increased *throughput*
  - Economy of scale
  - Increased reliability
    - graceful degradation
    - fault-tolerant systems
Parallel Systems (Cont.)

- **Symmetric multiprocessing (SMP)**
  - All processors are peers
  - Kernel routines can execute on different CPUs, in parallel

- **Asymmetric multiprocessing**
  - Master/slave structure
  - The kernel runs on a particular processor
  - Other CPUs can execute user programs and OS utilities.
Multi-core architectures
- Include multiple computing cores on a single chip
- Faster and more energy-efficient than multiple chips with single cores
Processes

- Process Concept
- Process States
- Process Creation and Termination
- Process Scheduling
- Process Communication
Process Concept

- **Process**: a program in execution
  - process execution must progress in sequential fashion.
- A program is a passive entity, whereas a process is an active entity with a program counter and a set of associated resources.
Each process has its own address space:
- Text section (text segment) contains the executable code
- Data section (data segment) contains the global variables
- Stack contains temporary data (local variables, return addresses..)

A process may contain a heap, which contains memory that is dynamically allocated at run-time.

The program counter and CPU registers are part of the process context.
CPU Switch From Process to Process

Diagram showing the process of a CPU switch from one process to another. The diagram includes states such as 'executing', 'idle', and actions like 'interrupt or system call'. It also shows steps such as saving state into PCB, reloading state from PCB, and an operating system in the middle.
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.

- Context-switch time is pure overhead; the system does no useful work while switching.

- Overhead dependent on hardware support (typically 1-1000 microseconds).
Multithreading

- When a multithreaded process is run on a single CPU system, the threads take turns running.
- All threads in the process have exactly the same address space.

**Per Process Items**
- Address Space
- Global Variables
- Open Files
- Accounting Information

**Per Thread Items**
- Program Counter
- Registers
- Stack
- State
Process Communication

Mechanism for processes to communicate and to synchronize their actions.

- Two models:
  - Communication through a shared memory region
  - Communication through message passing
Process Synchronization

- Race Conditions
- The Critical Section Problem
- Synchronization Hardware
- Semaphores
- Classical Problems of Synchronization
- Monitors
Suppose that two processes A and B have access to a shared variable “Balance”:

**PROCESS A:**
Balance = Balance - 100

**PROCESS B:**
Balance = Balance - 200

Further, assume that Process A and Process B are executing concurrently in a time-shared, multi-programmed system.
CPU Scheduling

- Basic Concepts

- Scheduling Criteria

- Scheduling Algorithms
  - First-Come-First-Served
  - Shortest-Job-First, Shortest-remaining-Time-First
  - Priority Scheduling
  - Round Robin
  - Multi-level Queue
  - Multi-level Feedback Queue
Non-preemptive vs. Preemptive Scheduling

- Under **non-preemptive scheduling**, each running process keeps the CPU until it completes or it switches to the waiting (blocked) state.

- Under **preemptive scheduling**, a running process may be also forced to release the CPU even though it is neither completed nor blocked.
  - In time-sharing systems, when the running process reaches the end of its time **quantum (slice)**
  - In general, whenever there is a change in the ready queue.
Starvation

- **Starvation** occurs when a job cannot make progress because some other job has the resource it requires.
  - We've seen locks, Monitors, Semaphores, etc.
  - The same thing can happen with the CPU!

- **Starvation** can be a side effect of synchronization.
  - Constant supply of readers always blocks out writers.
  - Well-written critical sections should ensure bounded waiting.

- **Starvation** is usually a side effect of the scheduling algorithm:
  - A high priority process always prevents a low priority process from running on the CPU.
  - One thread always beats another when acquiring a lock.
Multilevel Feedback Queue

- Multilevel feedback queue scheduler is 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

- The scheduler can be configured to match the requirements of a specific system.
Deadlock Definition

- **Deadlock** is a problem that can arise:
  - When processes compete for access to limited resources
  - When processes are incorrectly synchronized

- **Definition:**
  - Deadlock exists among a set of processes if every process is waiting for an event that can be caused only by another process in the set.

![Diagram of processes](image-url)
**Conditions for Deadlock**

- **Deadlocks** can exist if and only if four conditions hold:

1) **Mutual exclusion:** At least one resource must be held in a non-sharable mode. *(I.e., only one instance)*

2) **Hold and wait:** There must be one process holding one resource and waiting for another resource

3) **No preemption:** Resources cannot be preempted *(I.e., critical sections cannot be aborted externally)*

4) **Circular wait:** There must exist a set of processes \( \{P_1, P_2, P_3, \ldots, P_n\} \) such that \( P_1 \) is waiting for a resource held by \( P_2 \), \( P_2 \) is waiting for \( P_3 \), \ldots , and \( P_n \) for \( P_1 \)
Deadlock can be described using a resource allocation graph (RAG).

The RAG consists of sets of vertices $P = \{P_1, P_2, \ldots, P_n\}$ of processes and $R = \{R_1, R_2, \ldots, R_m\}$ resources.

- A directed edge from a process to a resource, $P_i \rightarrow R_j$, implies that $P_i$ has requested $R_j$.
- A directed edge from a resource to a process, $R_j \rightarrow P_i$, implies that $R_j$ has been acquired by $P_i$.
- Each resource has a fixed number of units.

If the graph has no cycles, deadlock cannot exist.

If the graph has a cycle, deadlock may exist.
Deadlock Avoidance

**SAFE:** R1 is assigned to P1 with priority

**UNSAFE:** (P1 requests R2) -> cycle
Deadlock Detection

- **Detection**
  - Traverse the resource graph looking for cycles
  - If a cycle is found, preempt resource (force a process to release)

- **Expensive**
  - Many processes and resources to traverse

- **Only invoke detection algorithm depending on**
  - How often or likely deadlock is
  - How many processes are likely to be affected when it occurs
Deadlock Summary

- Deadlock occurs when processes are waiting on each other and cannot make progress
  - Cycles in Resource Allocation Graph (RAG)

- Deadlock requires four conditions
  - Mutual exclusion, hold and wait, no resource preemption, circular wait

- Four approaches to dealing with deadlock:
  - Ignore it – Living life on the edge
  - Prevention – Make one of the four conditions impossible
  - Avoidance – Banker’s Algorithm (control allocation)
  - Detection and Recovery – Look for a cycle, preempt or abort