Review

- Computer Overview
- OS Overview
- OS Structures
- Process and Process Control
- Thread
- Scheduling
- Concurrent Programming and Mutual Exclusion

Computer Systems

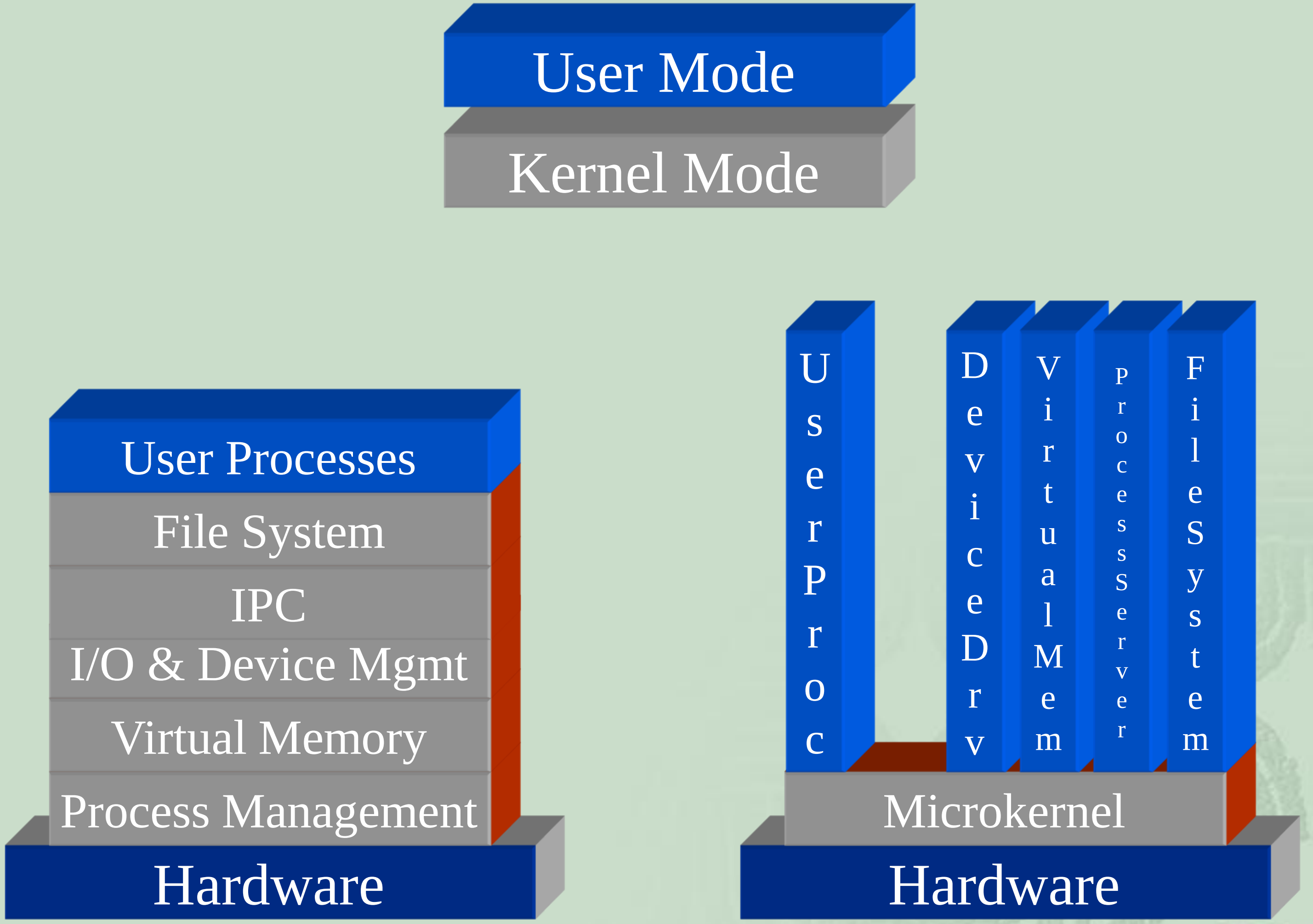
- Registers
- Interrupts
- Caching
- Memory Hierarchy
- Input/output
- Protection

Operating Systems

- Objectives: Convenience, Efficiency and Ability to evolve.
  - Process and Thread
  - Scheduling
  - Concurrency and Mutual exclusion
  - Resource management
  - Memory Management
  - File System
  - Communication

OS Structures

- Layered Structure
- Multi-kernels for Multi-core processors
- Microkernel Structure (IPC)
- Virtual Machine Structure



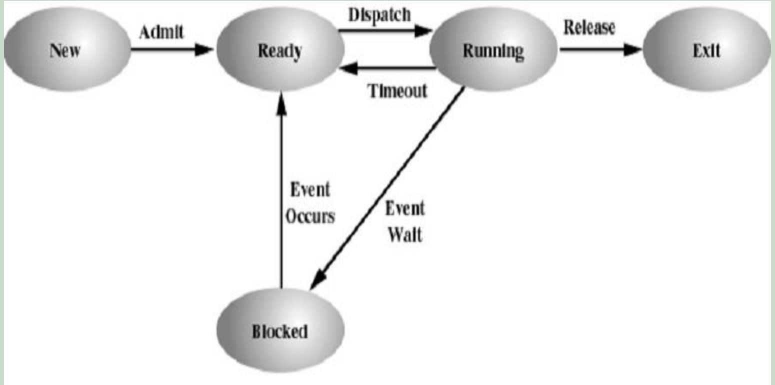
Traditional OS

Microkernel OS

Process Concepts

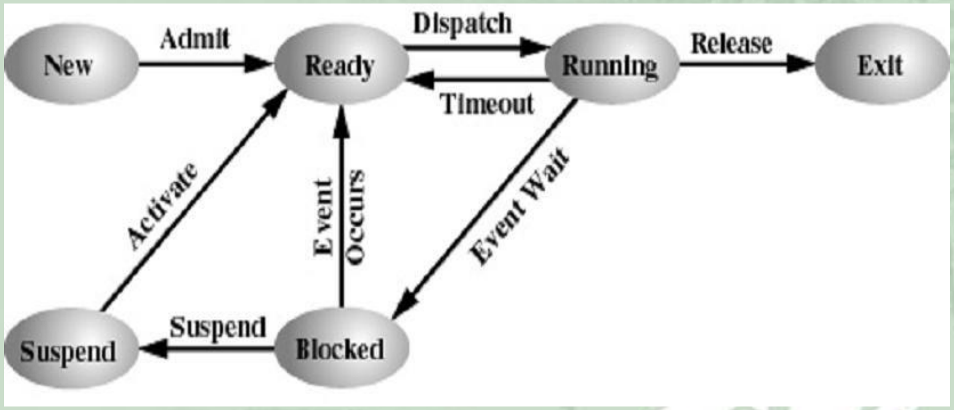
- Process Definition
- Process Creation
- Process States and State Transitions
- Important Information Associated with Processes (PCB)

State Transitions



Five States

+Suspend State



Process Control

- Process Control Block (PCB)
- Process Execution
- Context Switch
- Process Scheduling

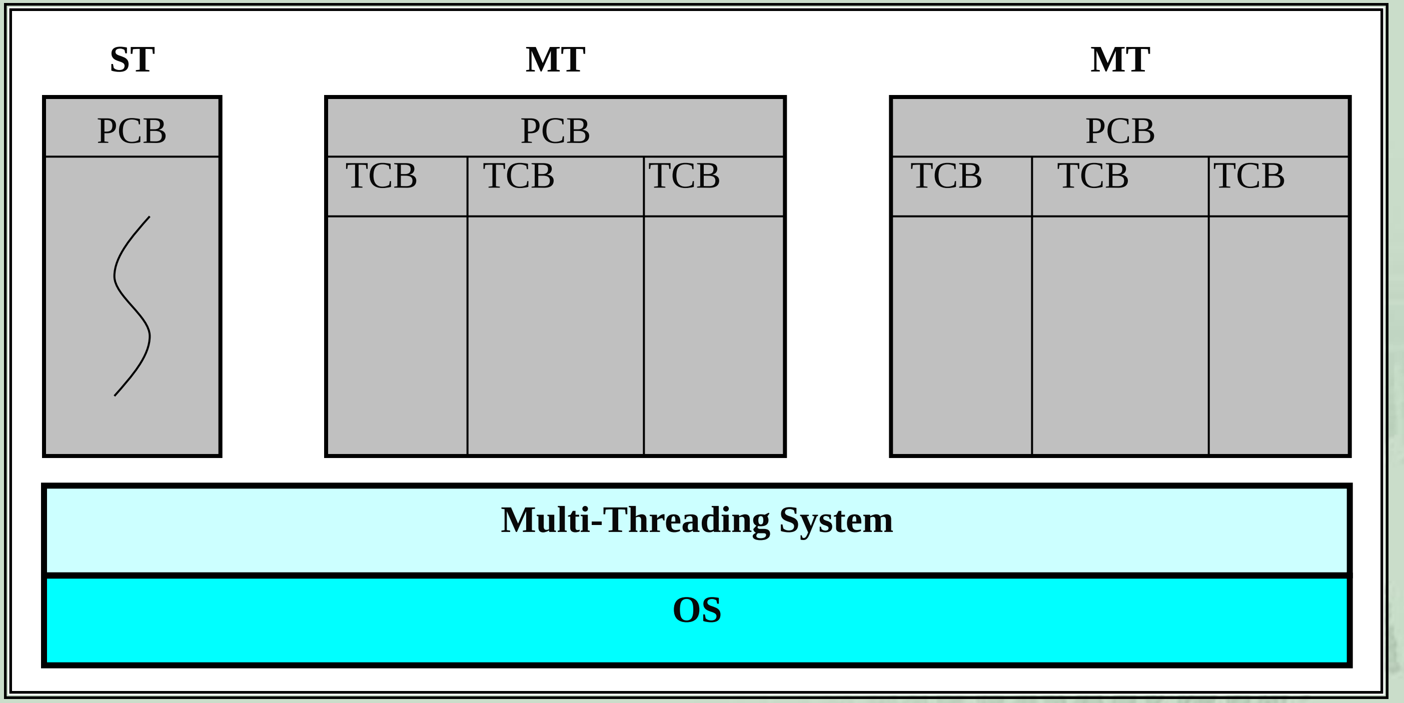
When to Switch a Process

- Clock interrupt
- I/O interrupt
- Memory fault
- Trap (error occurred)
- Supervisor call (system calls)
- Context Switch:
  - Saving the state of one process and loading another process onto the CPU

Threads

- A thread consists of:
  - state (Running, Ready, etc.)
  - CPU context (program counter, register set.)
  - an execution stack.
  - access to the memory and resources of its process (shared with all other threads in that process.)
  - OS resources (open files, signals, etc.)
- Benefits:
  - Far less time to create/terminate.
  - Switching between threads is faster.
  - No memory management issues, etc.

Threads and Processes



PCB : Process Control Block
TCB : Thread Control Block
contains information and status of thread/process

Scheduling

- Objectives:
  - minimize response time
  - maximize throughput
  - maximize processor efficiency
- Long-term
  - performed when new process is created
- Medium-term
  - swapping to maintain a degree of multiprogramming
- Short-term
  - which ready process to execute next – dispatcher

Preemptive vs. Non-preemptive

- Scheduling that only takes place due to I/O or process termination is non-preemptive.
- Preemptive scheduling allows the operating system to interrupt the currently running process and move it to the ready state.
- Preemptive scheduling:
  - incurs greater overhead (context switch)
  - provides better service to the total population of processes
  - may prevent one process from monopolizing the processor

Algorithms

- First Come First Served
  - Processes queued in order of arrival
  - Runs until finished or blocks on I/O
- Round Robin
  - FCFS with preemption
  - Size of ticks affects performance
- Shortest Process Next
  - Select process with shortest expected running time (non-preemptive)
  - Difficult to estimate required time

Algorithms

- Highest Response Ratio Next
  - Non-preemptive, tries to get best average normalized turnaround time
  - Depends on Response Ratio
    - W = time spent waiting
    - S = expected service time

$$RR = (W + S) / S$$

- Priority
  - Schedule Process with the highest priority
- Feedback Queue
  - Starts in high-priority queue, moves down in priority as it executes
  - Lower-priority queues often given longer time slices

FCFS

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive at time 0 in the order: P_1 , P_2 , P_3
  The simplified Gantt Chart for the schedule is:

- Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time: $(0 + 24 + 27)/3 = 17$

RR with Time Quantum = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- The Gantt chart is:

P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_3

0 20 37 57 77 97 117 121 134 154 162

Resource Competition

- Mutual Exclusion
  - Critical resource – a single non-sharable resource.
  - Critical section – portion of the program that accesses a critical resource.
- Deadlock
  - Each process owns a resource that the other is waiting for.
  - Two processes are waiting for communication from the other.
- Starvation
  - A process is denied access to a resource, even though there is no deadlock situation.

Mutual Exclusion and Synchronization

- Software Solutions:
  - Shared variables, take turns
- Hardware Solutions:
  - Special Instructions: test&set, swap
- OS Solutions:
  - Semaphores: Binary or Counting
  - Operations: P/V, wait/signal, lock/unlock
  - Monitors

A simple synchronization example

Expression: (x*2) / (y-z) + sin(x)

Shared variables: t1, t2, t3, t4, t5;

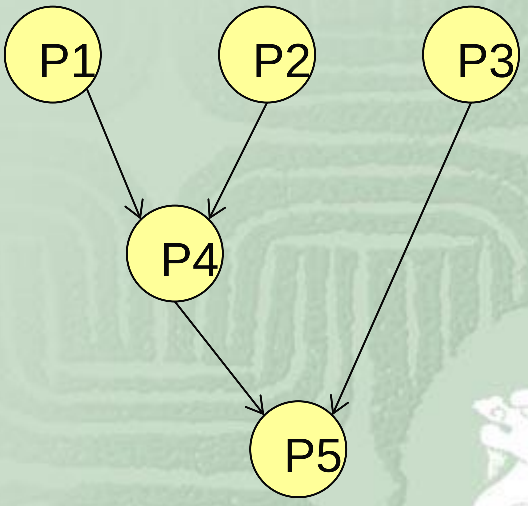
P1: t1 = x*2;

P2: t2 = y – z;

P3: t3 = sin(X);

P4: t4 = t1/t2;

P5: t5 = t4 + t3



Concurrency control

semaphore s1, s2; // initial 0s

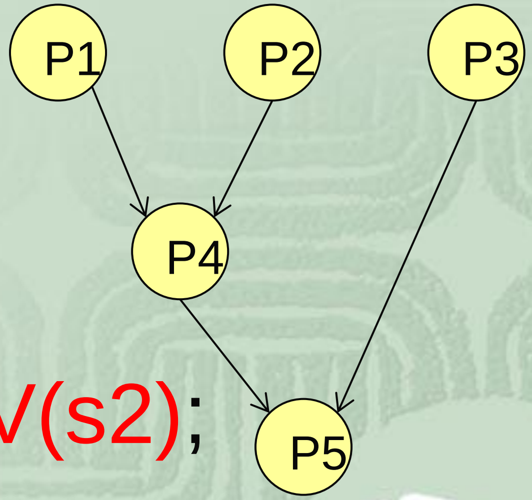
P1: t1 = x*2; V(s1);

P2: t2 = y – z; V(s1);

P3: t3 = sin(X); V(s2);

P4: P(s1); P(s1); t4 = t1/t2; V(s2);

P5: P(s2); P(s2); t5 = t4 + t3;



Bounded Buffer Solution

```
Shared semaphore: empty = n, full = 0, mutex = 1;
Item buffer[n], int in = out = 0;
```

Producer

```
repeat
  produce an item in nextp

  wait(empty);
  wait(mutex);

  buffer[in] = nextp
  in = (in + 1) mod n

  signal(mutex);
  signal(full);

until false
```

Consumer

```
repeat
  wait(full);
  wait(mutex);

  nextc = buffer[out]
  out = (out + 1) mod n

  signal(mutex);
  signal(empty);

  consume the item in nextc

until false
```

Mutex + Synchronization: multiple producers and comusers