Process management

What are we going to learn?

- ***Processes :*** Concept of processes, process scheduling, co-operating processes, inter-process communication.

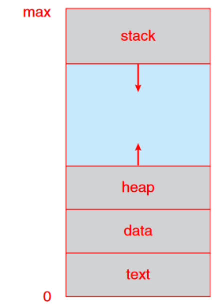
- ***CPU scheduling :*** scheduling criteria, preemptive & non-preemptive scheduling, scheduling algorithms (FCFS, SJF, RR, priority), algorithm evaluation, multi-processor scheduling.

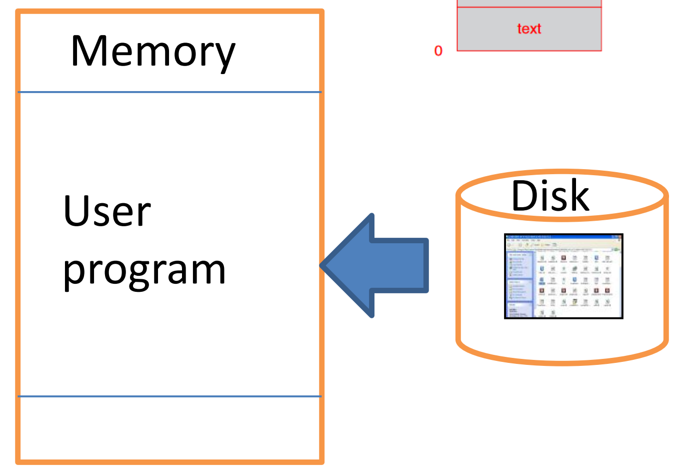
- ***Process Synchronization :*** background, critical section problem, critical region, synchronization hardware, classical problems of synchronization, semaphores.

- ***Threads :*** overview, benefits of threads, user and kernel threads.

- ***Deadlocks :*** system model, deadlock characterization, methods for handling deadlocks, deadlock prevention, deadlock avoidance, deadlock detection, recovery from deadlock.

Process concept



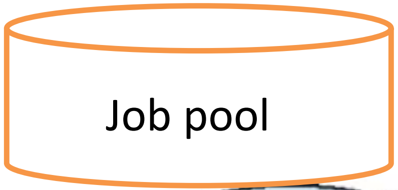


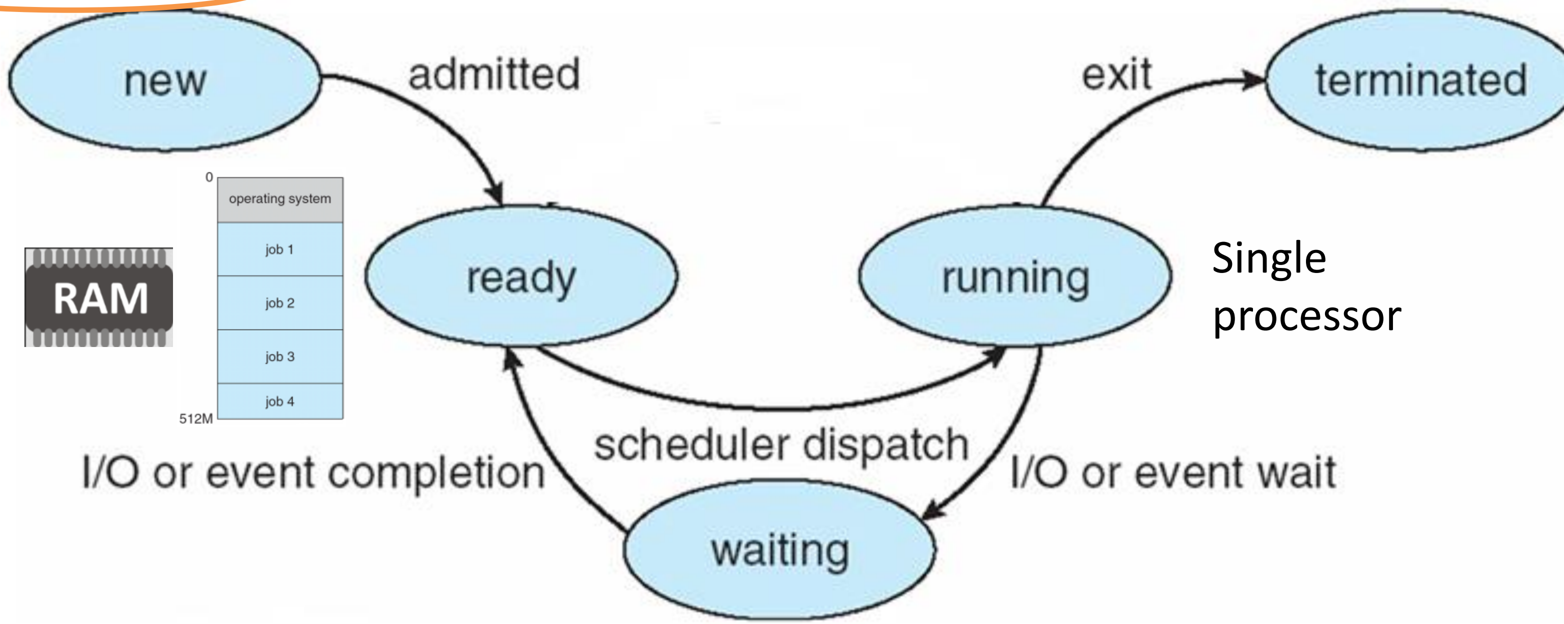
- Process is a dynamic entity
  - Program in execution
- Program code
  - Contains the text section
- Program becomes a process when
  - executable file is loaded in the memory
  - Allocation of various resources
    - Processor, register, memory, file, devices
- One program code may create several processes
  - One user opened several MS Word
  - Equivalent code/text section
  - Other resources may vary

Process State

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

Process State diagram



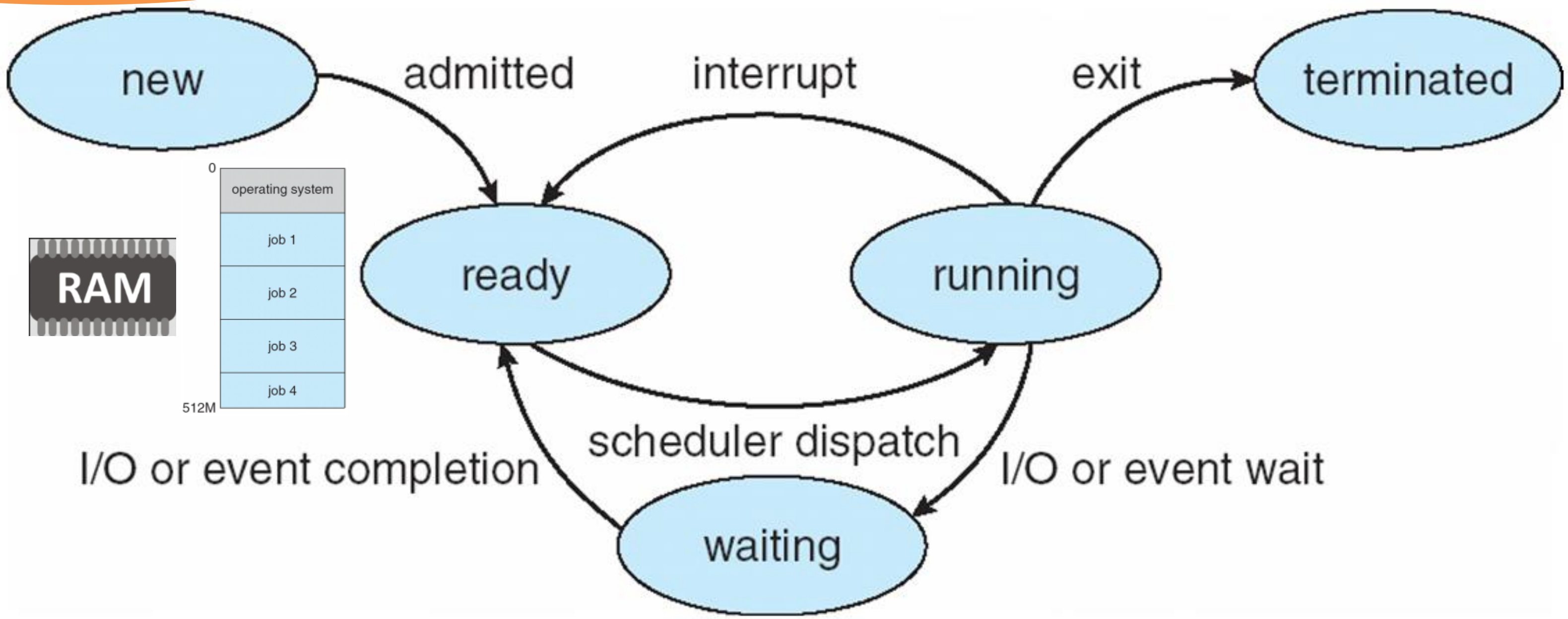


Multiprogramming

As a process executes, it changes *state*

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

Process State diagram



Multitasking/Time sharing

As a process executes, it changes *state*

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

How to represent a process?

- Process is a dynamic entity
  - Program in execution
- Program code
  - Contains the text section
- Program counter (PC)
- Values of different registers
  - Stack pointer (SP) (maintains process stack)
    - Return address, Function parameters
  - Program status word (PSW)

C	Z	O	S	I	K	

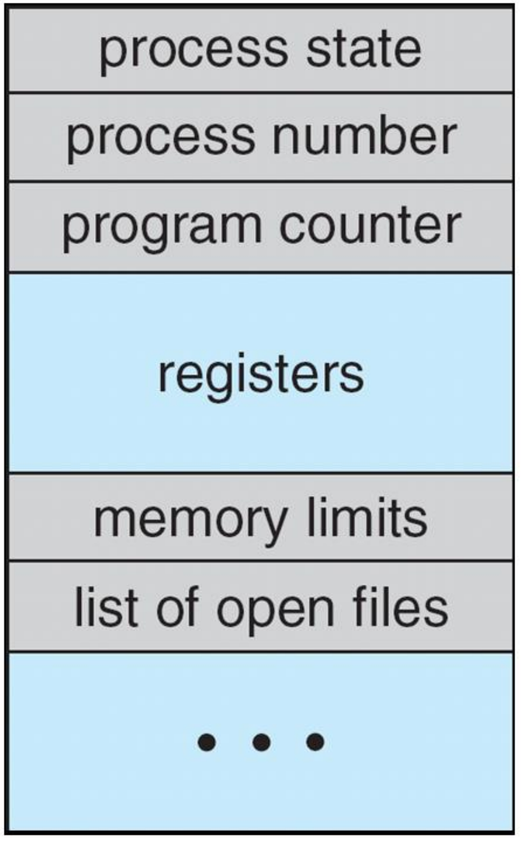
  - General purpose registers
- Main Memory allocation
  - Data section
    - Variables
  - Heap
    - Dynamic allocation of memory during process execution

Process Control Block (PCB)

- Process is represented in the operating system by a Process Control Block

Information associated with each process

- Process state
- Program counter
- CPU registers
  - Accumulator, Index reg., stack pointer, general Purpose reg., Program Status Word (PSW)
- CPU scheduling information
  - Priority info, pointer to scheduling queue
- Memory-management information
  - Memory information of a process
  - Base register, Limit register, page table, segment table
- Accounting information
  - CPU usage time, Process ID, Time slice
- I/O status information
  - List of open files=> file descriptors
  - Allocated devices

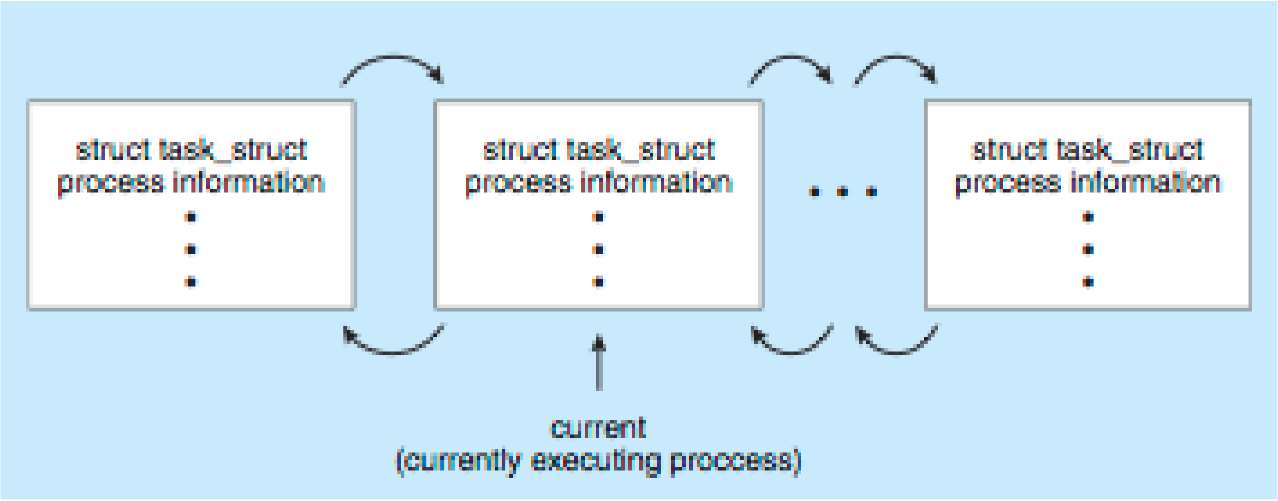


Process Representation in Linux

Represented by the C structure `task_struct`

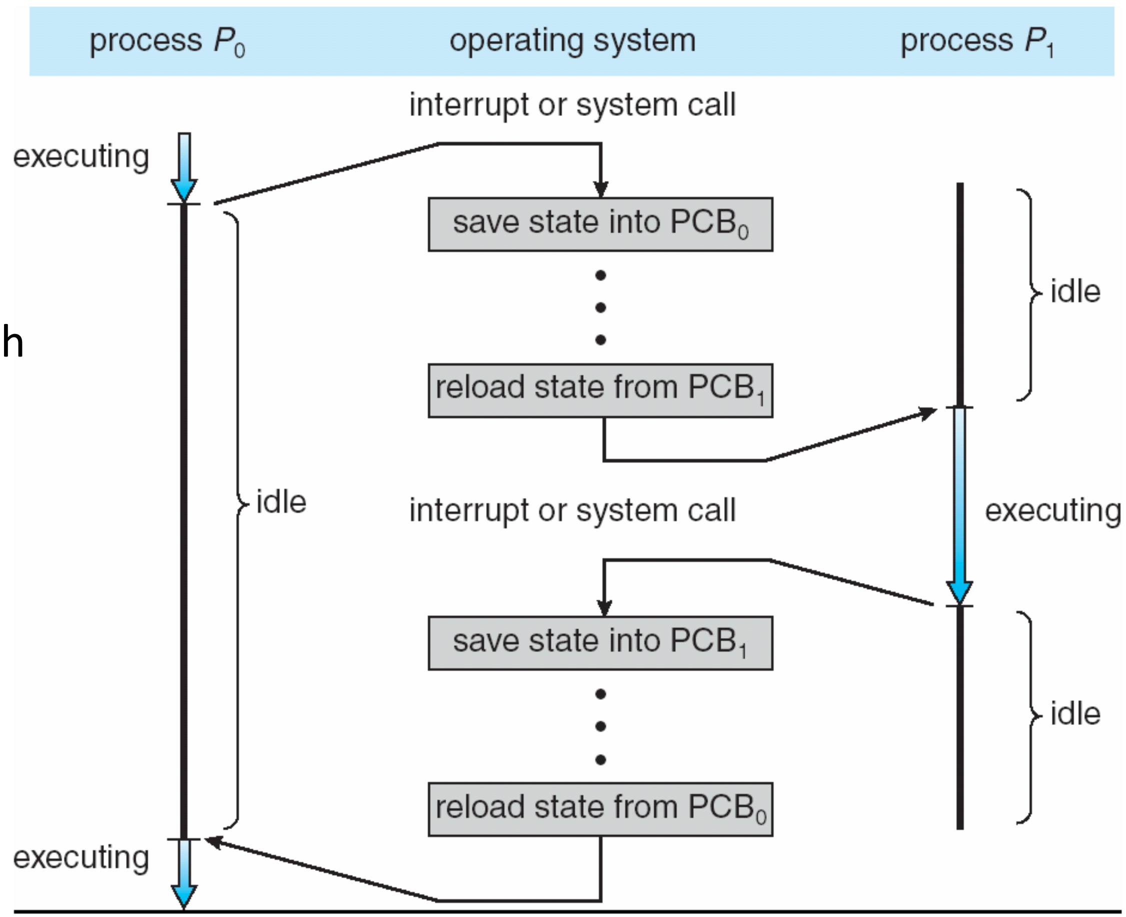
```
pid t pid; /* process identifier */
long state; /* state of the process */
unsigned int time slice /* scheduling information */
struct task struct *parent; /* this process's parent */
struct list head children; /* this process's children */
struct files struct *files; /* list of open files */
struct mm_struct *mm; /* address space of this pro */
```

Doubly linked list

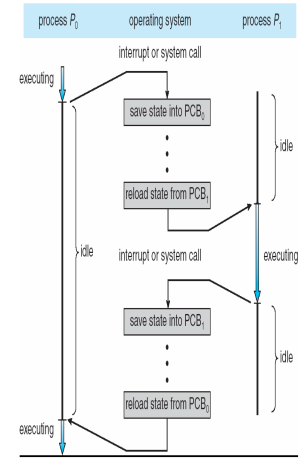


CPU Switch From Process to Process

Context switch



Context Switch



- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a **context switch**.

- **Context** of a process represented in the PCB

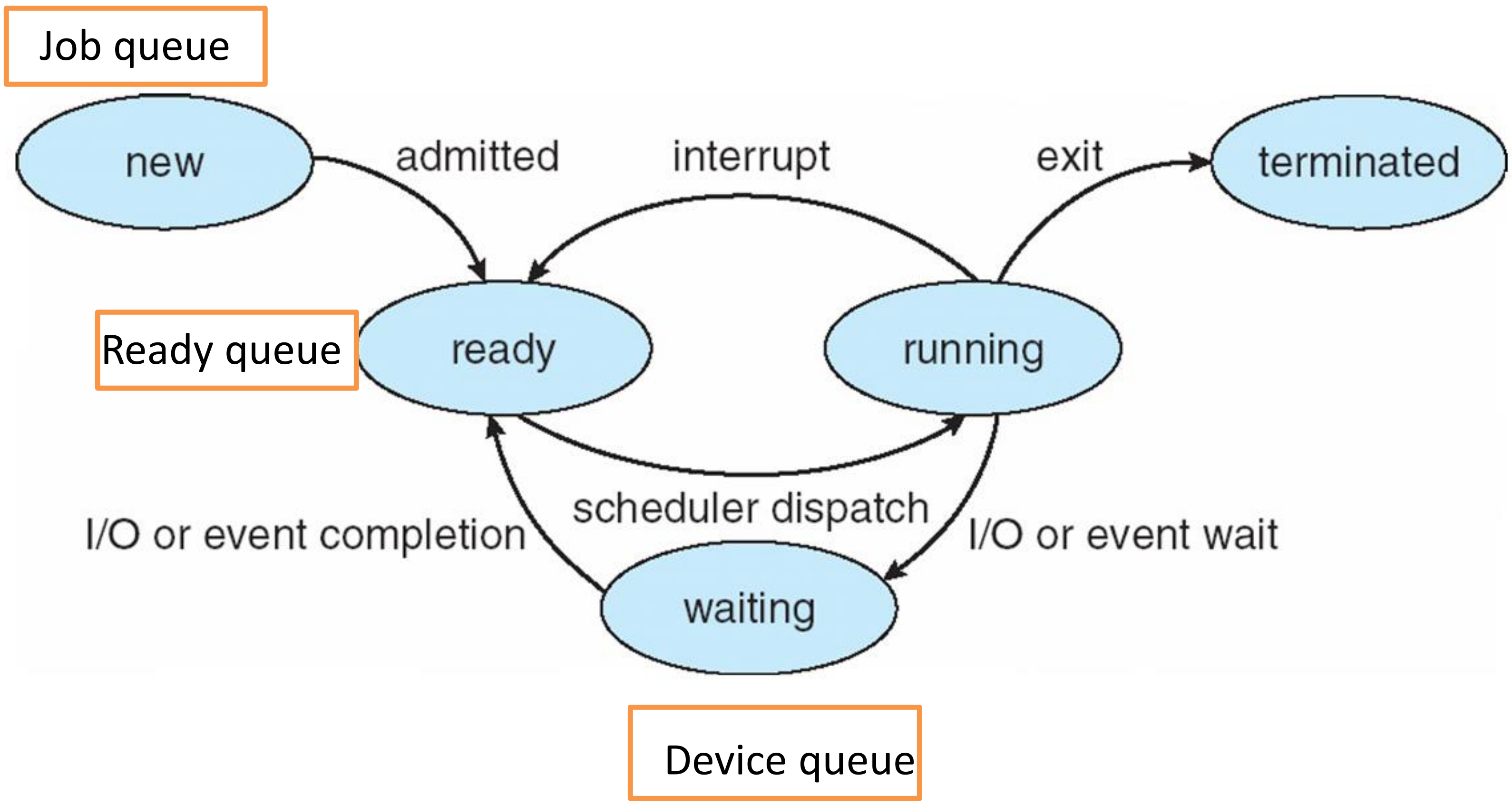
- Context-switch time is overhead; the system does not do useful work while switching
  - The more complex the OS and the PCB -> longer the context switch

- Time dependent on hardware support
  - Some hardware provides multiple sets of registers per CPU -> multiple contexts loaded at once

Scheduling queues

- Maintains **scheduling queues** of processes
  - **Job queue** – set of all processes in the system
  - **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
  - **Device queues** – set of processes waiting for an I/O device
- Processes migrate among the various queues

Scheduling queues

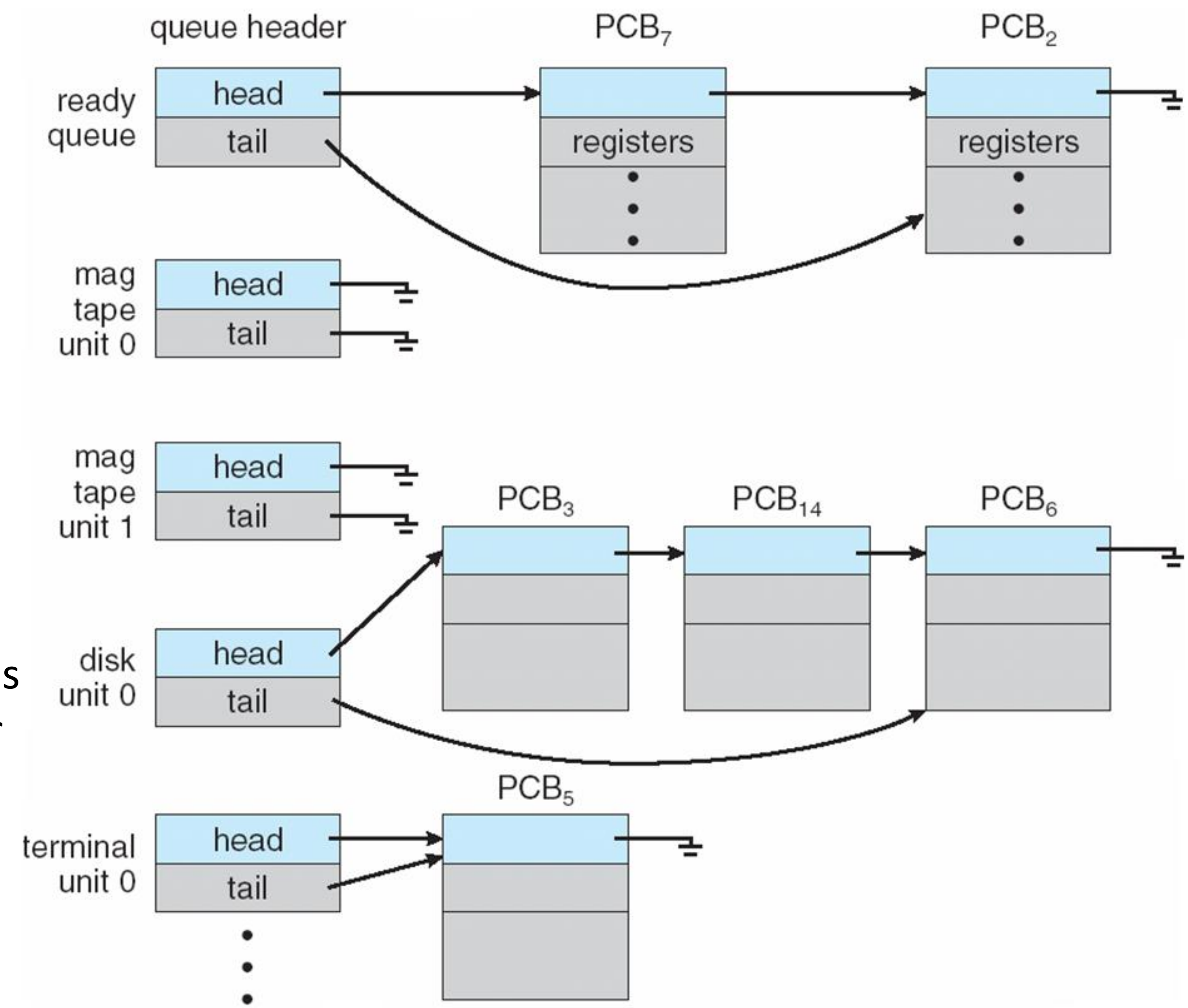


Ready Queue And Various I/O Device Queues

Queues are linked list of PCB's

Device queue

Many processes are waiting for disk



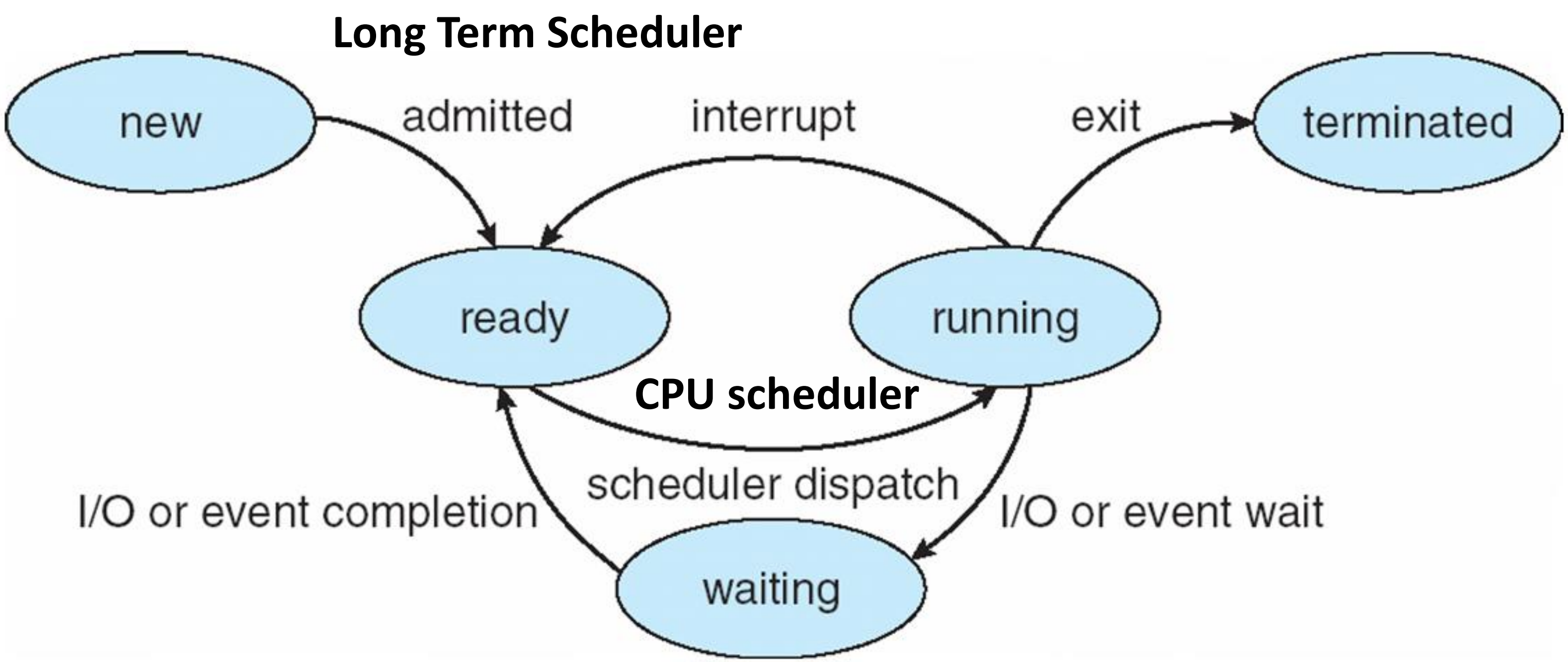
Process Scheduling

- We have various queues
- Single processor system
  - Only one CPU=> only one running process
- Selection of one process from a group of processes
  - **Process scheduling**

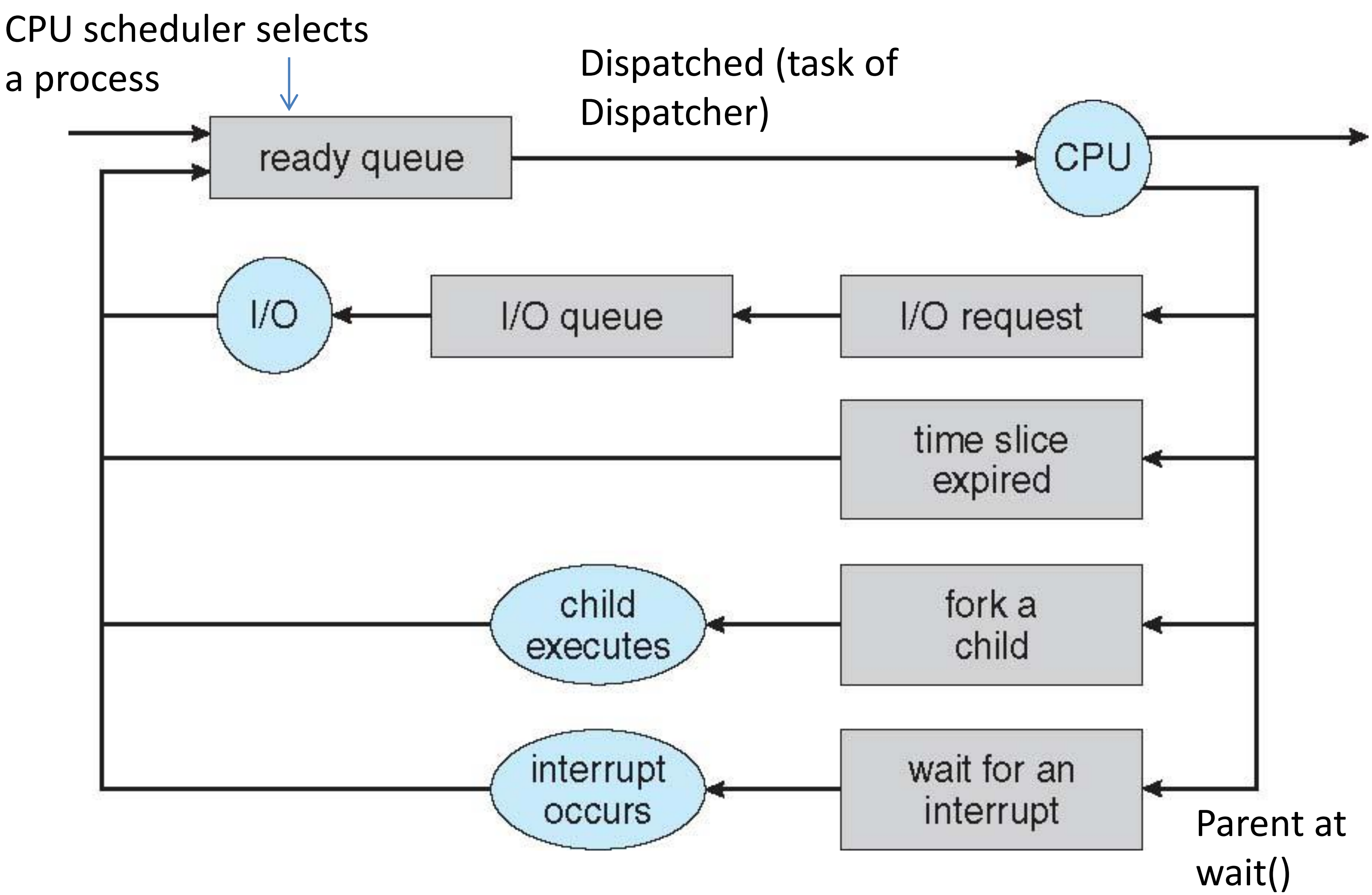
Process Scheduling

- Scheduler
  - Selects a process from a set of processes
- Two kinds of schedulers

1. Long term schedulers, job scheduler
   - A large number of processes are submitted (more than memory capacity)
   - Stored in disk
   - Long term scheduler selects process from job pool and loads in memory
2. Short term scheduler, CPU scheduler
   - Selects one process among the processes in the memory (ready queue)
   - Allocates to CPU


Long Term Scheduler
new
admitted
interrupt
exit
terminated
ready
running
CPU scheduler
scheduler dispatch
I/O or event completion
I/O or event wait
waiting

Representation of Process Scheduling

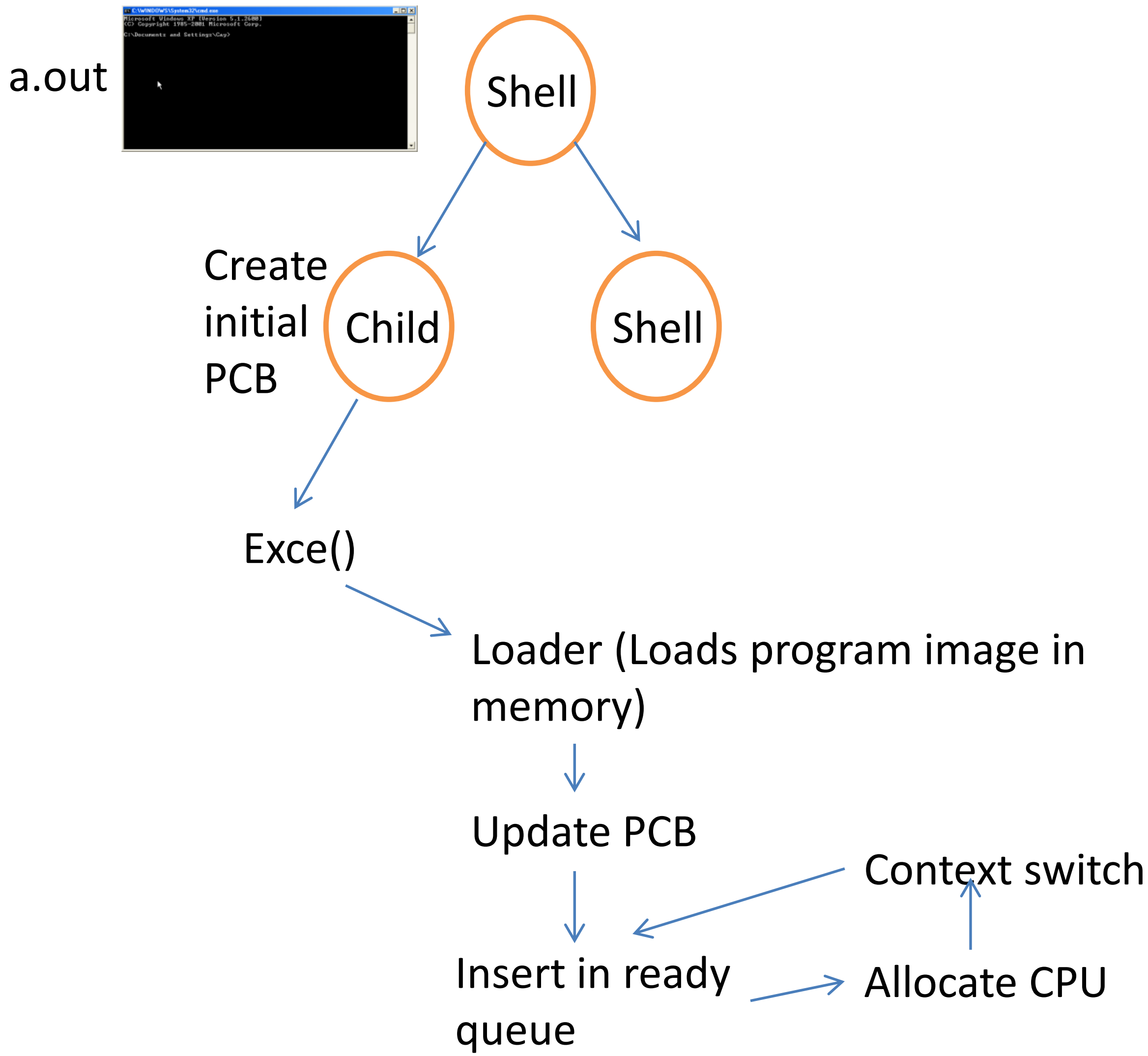


Dispatcher

- 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

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

Creation of PCB



Schedulers

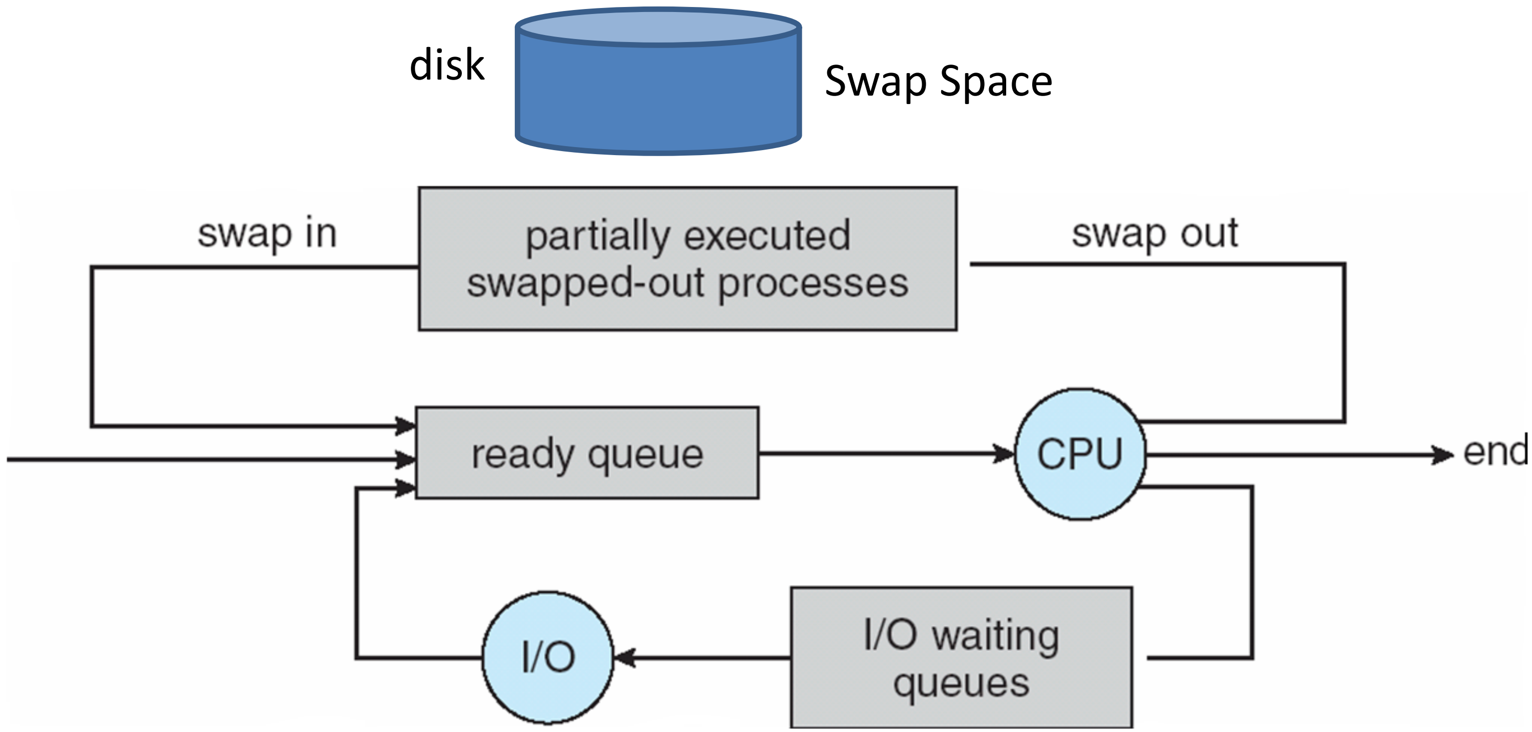
- **Scheduler**
  - Selects a process from a set
- **Long-term scheduler** (or job scheduler) – selects which processes should be brought into the ready queue
- **Short-term scheduler** (or CPU scheduler) – selects which process should be executed next and allocates CPU
  - Sometimes the only scheduler in a system

Schedulers: frequency of execution

- Short-term scheduler is invoked **very frequently** (milliseconds) $\Rightarrow$ (must be fast)
  - After a I/O request/ Interrupt
- Long-term scheduler is invoked very infrequently (seconds, minutes) $\Rightarrow$ (may be slow)
  - The long-term scheduler controls the *degree of multiprogramming*

- Processes can be described as either:
  - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
    - Ready queue empty
  - **CPU-bound process** – spends more time doing computations; few very long CPU bursts
    - Devices unused
- Long term scheduler ensures good process mix of I/O and CPU bound processes.

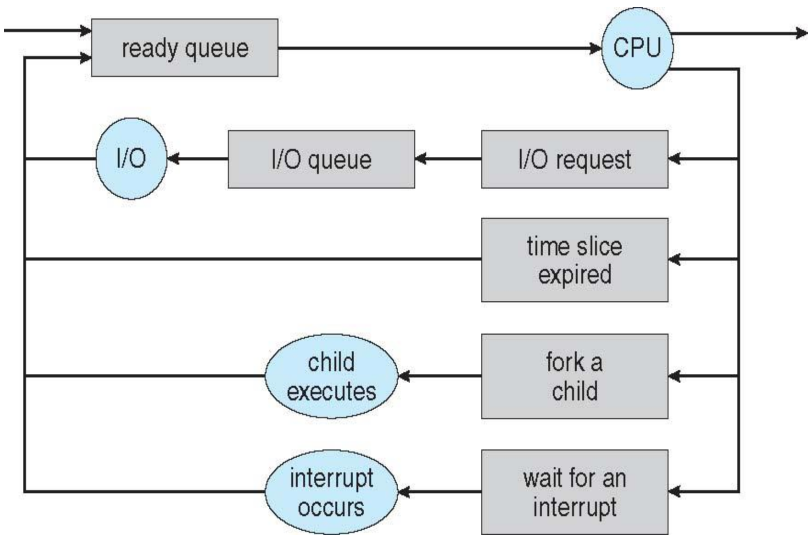
Addition of Medium Term Scheduling



Swapper

ISR for context switch

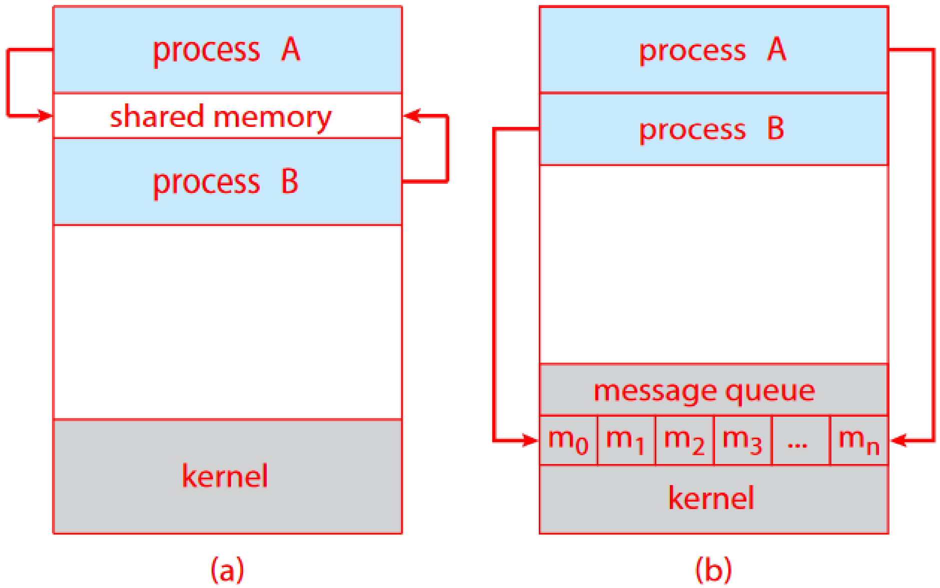
```
Current <- PCB of current process
Context_switch()
{
        Disable interrupt;
        switch to kernel mode
        Save_PCB(current);
        Insert(ready_queue, current);
        next=CPU_Scheduler(ready_queue);
        remove(ready_queue, next);
        Dispatcher(next);
        switch to user mode;
        Enable Interrupt;

}
Dispatcher(next)
{
        Load_PCB(next); [update PC]


}
```



Interprocess Communication

- Processes within a system may be **independent** or **cooperating**
- Cooperating process can affect or be affected by other processes, including sharing data
- Cooperating processes require an interprocess communication (IPC) mechanism that will allow them to exchange data— that is, send data to and receive data from each other.
- Cooperating processes need **interprocess communication** (**IPC**)
- Two models of IPC
  - Shared memory
  - Message passing

Interprocess Communication



In the **shared-memory model**, a **region of memory** that is shared by the cooperating processes is established.
Processes can then exchange information by reading and writing data to the shared region.

In the **message-passing model**, communication takes place by means of messages exchanged between the cooperating processes
(Kernel involvement, slow)

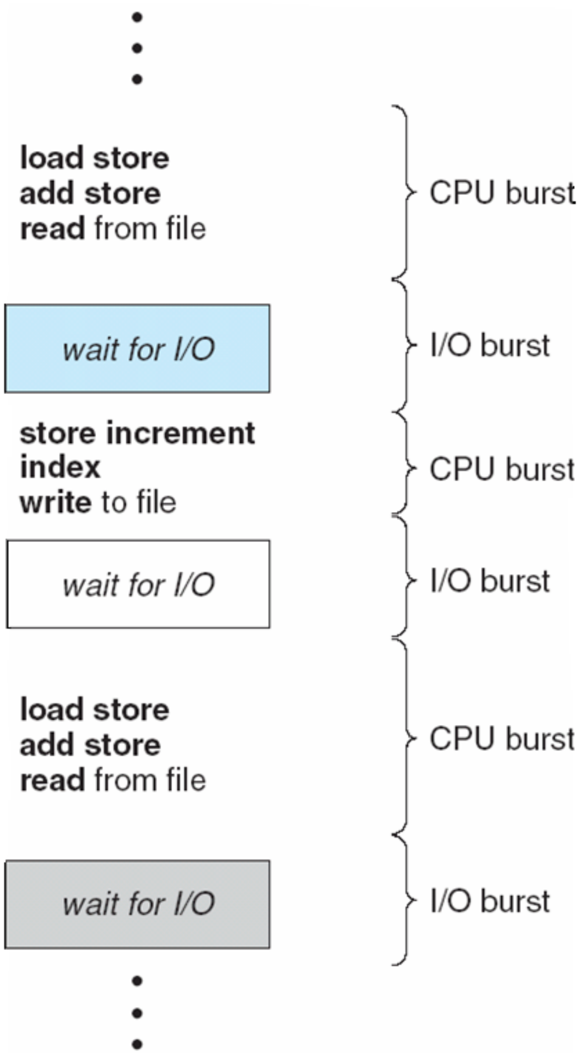
CPU Scheduling

- Describe various CPU-scheduling algorithms
- Evaluation criteria for selecting a CPU-scheduling algorithm for a particular system

Basic Concepts

- Maximum CPU utilization obtained with multiprogramming
  - Several processes in memory (ready queue)
  - When one process requests I/O, some other process gets the CPU
  - Select (schedule) a process and allocate CPU

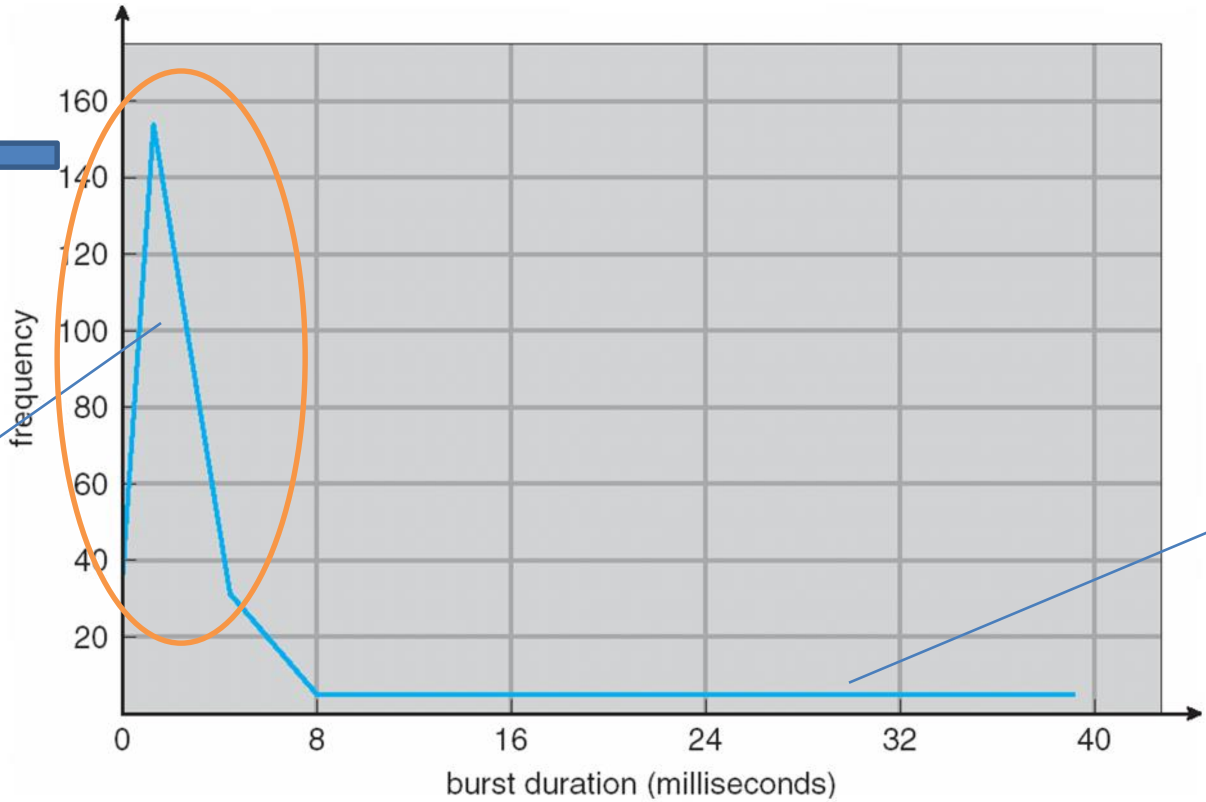
Observed properties of Processes



- CPU–I/O Burst Cycle
- Process execution consists of a *cycle* of CPU execution and I/O wait
- Study the duration of CPU bursts

Histogram of CPU-burst Times

Utility of CPU scheduler



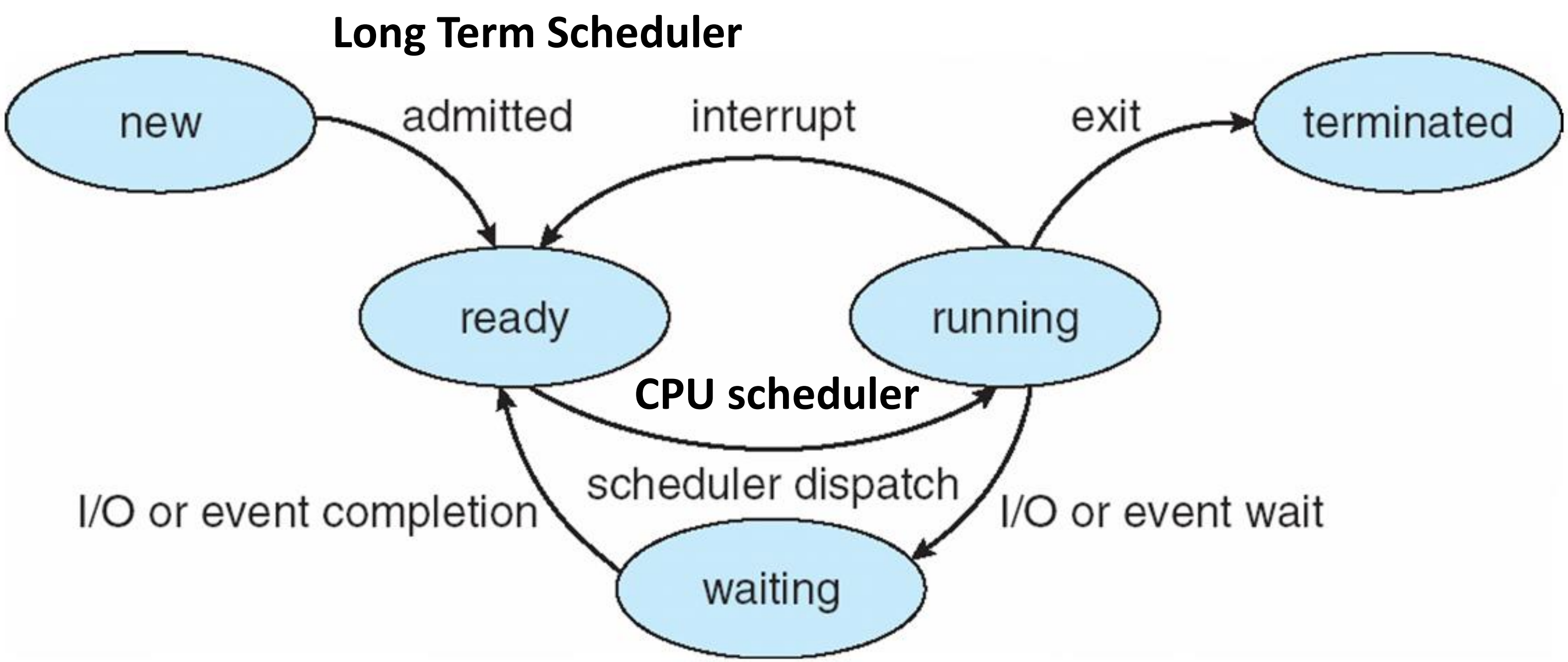
I/O bound process

CPU bound process

Large number of short CPU bursts and small number of long CPU bursts

Preemptive and non preemptive

- Selects from among the processes in ready queue, and allocates the CPU to one of them
  - Queue may be ordered in various ways (not necessarily FIFO)
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready
  4. Terminates
- Scheduling under 1 and 4 is **nonpreemptive**
- All other scheduling is **preemptive**



Long Term Scheduler
new
admitted
interrupt
exit
terminated
ready
running
CPU scheduler
scheduler dispatch
I/O or event completion
I/O or event wait
waiting

Preemptive scheduling

Preemptive scheduling

Results in cooperative processes

Issues:

– Consider access to shared data
  - Process synchronization

– Consider preemption while in kernel mode
  - Updating the ready or device queue
  - Preempted and running a “ps -el”

Race condition

Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit
- **Turnaround time** – amount of time to execute a particular process
- **Waiting time** – amount of time a process has been waiting in the ready queue
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output

Scheduling Algorithm Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

- Mostly optimize the average
- Sometimes optimize the minimum or maximum value
  - Minimize max response time

- For interactive system, variance is important
  - E.g. response time
- System must behave in predictable way

Scheduling algorithms

- First-Come, First-Served (FCFS) Scheduling
- Shortest-Job-First (SJF) Scheduling
- Priority Scheduling
- Round Robin (RR)

First-Come, First-Served (FCFS) Scheduling

- Process that requests CPU first, is allocated the CPU first
- Ready queue=>FIFO queue
- Non preemptive
- Simple to implement

Performance evaluation

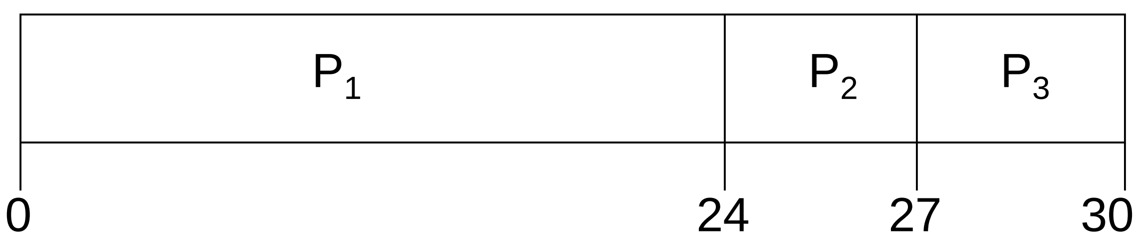
- Ideally many processes with several CPU and I/O bursts

- Here we consider only one CPU burst per process

First-Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

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



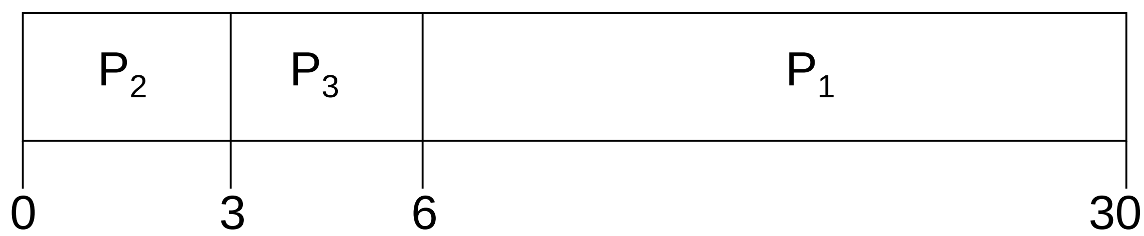
- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order:

$$P_2 , P_3 , P_1$$

- The Gantt chart for the schedule is:



- Waiting time for $P_1 = 6; P_2 = 0; P_3 = 3$
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case
- Average waiting time under FCFS heavily depends on process arrival time and burst time
- **Convoy effect** - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes

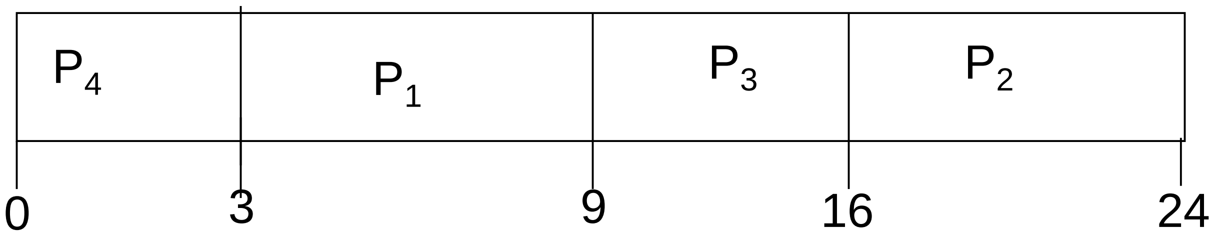
Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst
  - Allocate CPU to a process with the smallest next CPU burst.
  - Not on the total CPU time
- Tie=>FCFS

Example of SJF

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart



- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

Avg waiting time for FCFS?

SJF

- SJF is optimal – gives minimum average waiting time for a given set of processes (Proof: home work!)
- The difficulty is knowing the length of the next CPU request
- Useful for Long term scheduler
  - Batch system
  - Could ask the user to estimate
  - Too low value may result in “time-limit-exceeded error”

Preemptive version
Shortest-remaining-time-first

- Preemptive version called **shortest-remaining-time-first**
- Concepts of varying arrival times and preemption to the analysis

Process	*Arrival* Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- *Preemptive* SJF Gantt Chart

P_1	P_2	P_4	P_1	P_3

0	1	5	10	17	26

- Average waiting time = [(10-1)+(1-1)+(17-2)+(5-3)]/4 = 26/4 = 6.5 msec

Avg waiting time for non preemptive?

Determining Length of Next CPU Burst

- Estimation of the CPU burst length – should be similar to the previous burst
  - Then pick process with shortest predicted next CPU burst
- Estimation can be done by using the length of previous CPU bursts, using time series analysis

1. $t_n =$ actual length of n^{th} CPU burst
2. $\tau_{n+1} =$ predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define :

$$\tau_{n+1} = \alpha\ t_n + (1 - \alpha)\tau_n .$$

Boundary cases α=0, 1

- Commonly, α set to ½

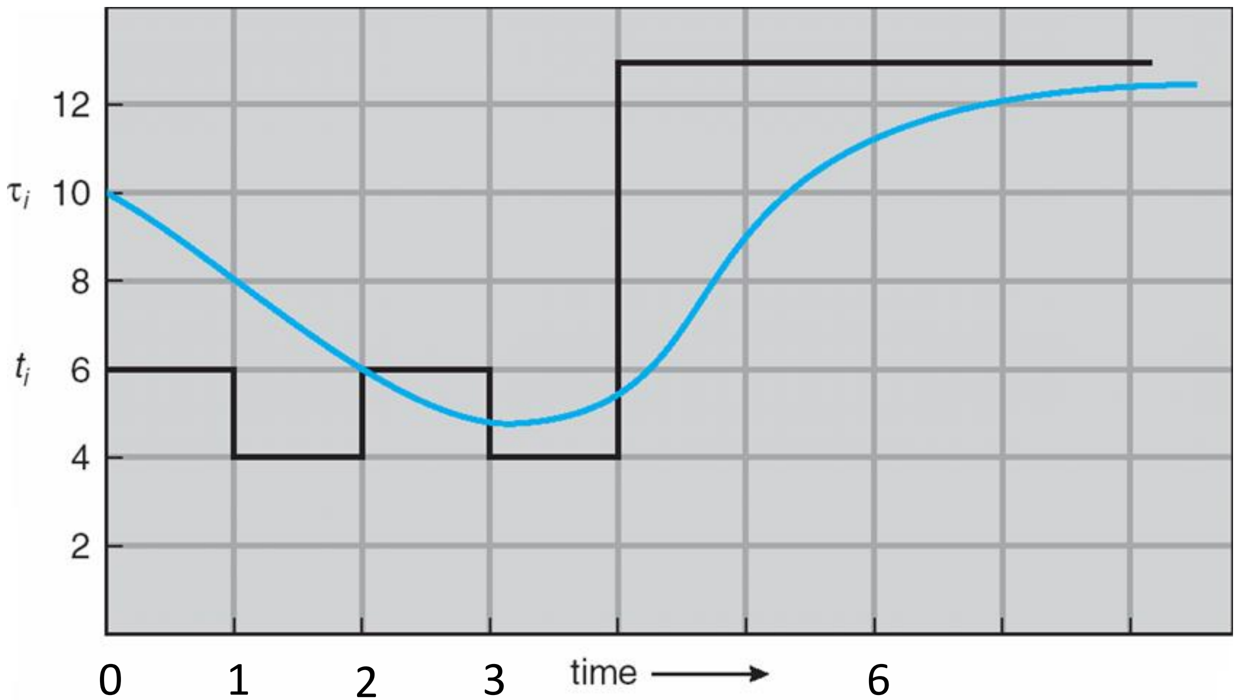
Examples of Exponential Averaging

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - Recent burst time does not count
- $\alpha = 1$
  - $\tau_{n+1} = t_n$
  - Only the actual last CPU burst counts
- If we expand the formula, we get:

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\alpha\, t_{n-1} + \dots$$
$$+ (1 - \alpha)^j \alpha\, t_{n-j} + \dots$$
$$+ (1 - \alpha)^{n+1} \tau_0$$

- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor

Prediction of the Length of the Next CPU Burst



CPU burst (t_i)		6	4	6	4	13	13	13	…
"guess" (τ_i)	10	8	6	6	5	9	11	12	…