Unit 2
Process Management and Scheduling

Dr.S.Thenmozhi

Contents

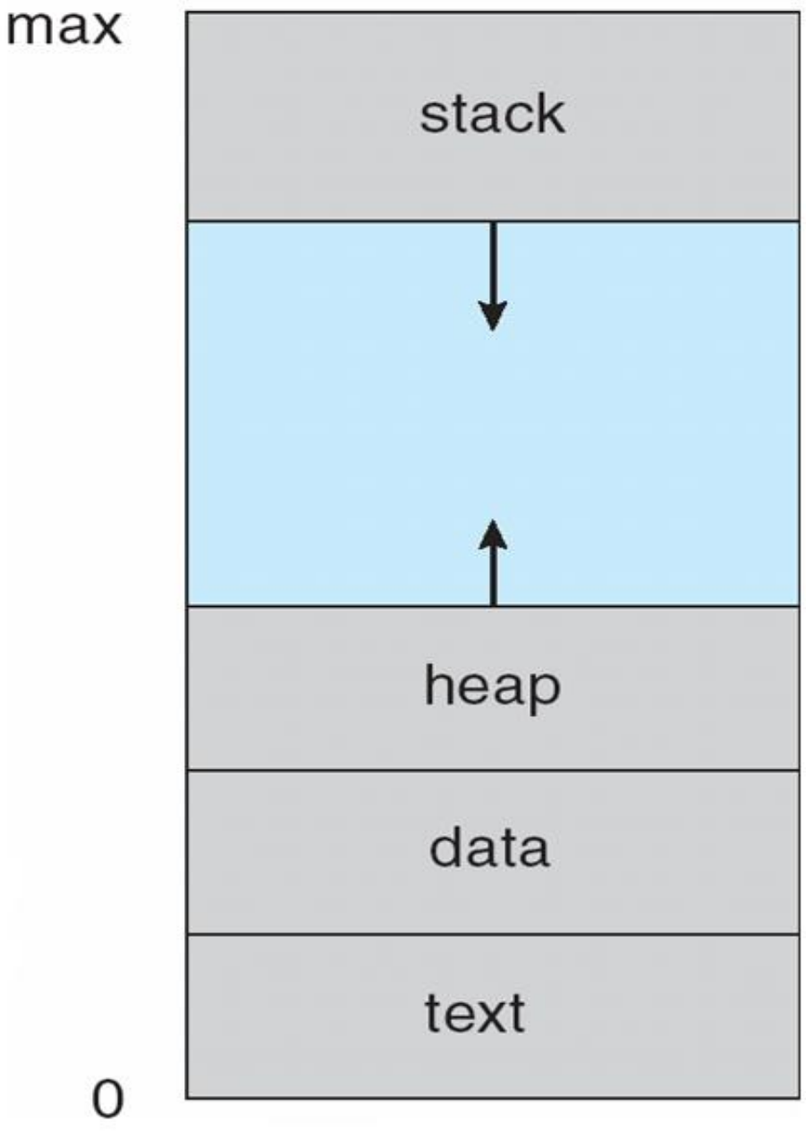
1. Process Concepts
2. Process Scheduling
3. Operations on Process
4. Inter- Process Communication
5. Threads
6. Multi Threading Models
7. Basic Concepts on Process Scheduling
8. Scheduling Criteria
9. Scheduling Algorithms
10. Algorithm Evaluation

2.1 Process Concept

- Process - An instance of **computer program in execution**
- Some times also called as jobs or task
  - Batch system – **jobs**
  - Time-shared systems – **user programs or tasks**
- Process on modern systems can have several threads of execution or several different pieces of the program running in parallel
- Each process has its own private resources, data and associated statistics
- A OS creates a data structure in the memory

Process Concepts – The Process

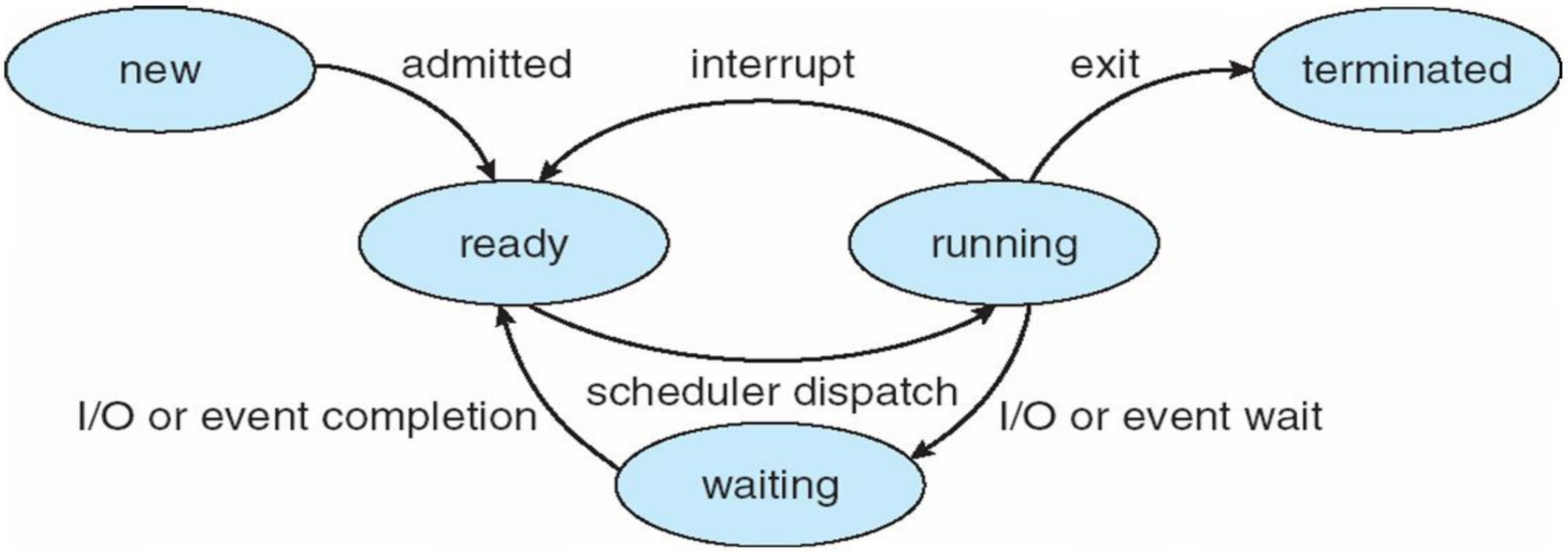
- In memory, a process consists of **multiple parts:**
  - **Program code**, also called **text section**
  - **Stack** containing temporary data
    - Function parameters, return addresses, local variables
  - **Data section** containing global variables
  - **Heap** containing memory dynamically allocated during run time



Process Concept - Process State

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

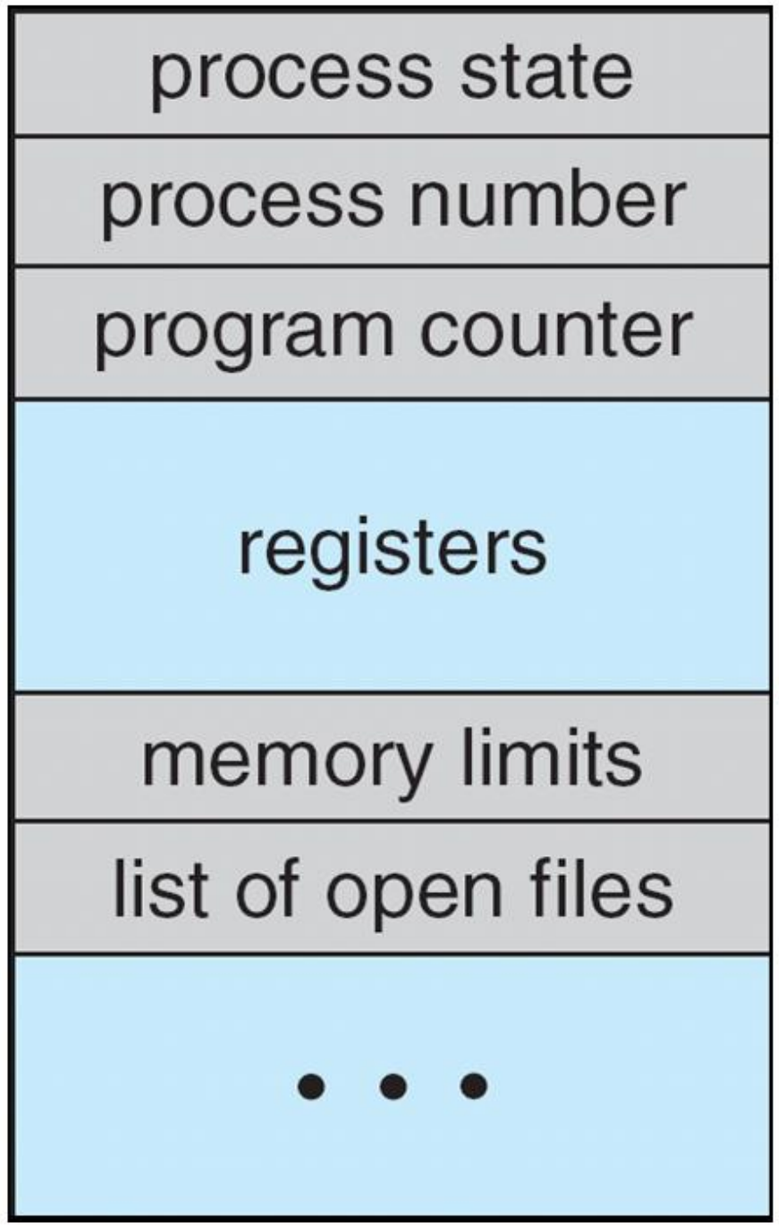
Diagram of Process State



Process Concepts - Process Control Block (PCB)

Information associated with each process

- Process state – new, ready, running…
- Process Number – process id, parent proces
- Program counter – next instr. for exec
- CPU registers – no. and type of reg.used
- Memory-management information
  - Limit of page table and segment table
- Accounting information
  - Amt of CPU time used, time limit
- I/O status information
  - List of I/O devices



Process Concepts - Threads

- Threads – lightweight process
- So far, process has a single thread of execution
  - One task at a time
- Consider having multiple program counters per process
  - Multiple locations can execute at once
  - Multiple tasks at a time
- PCB must be extended to handle threads:
  - Store thread details
  - Multiple program counters

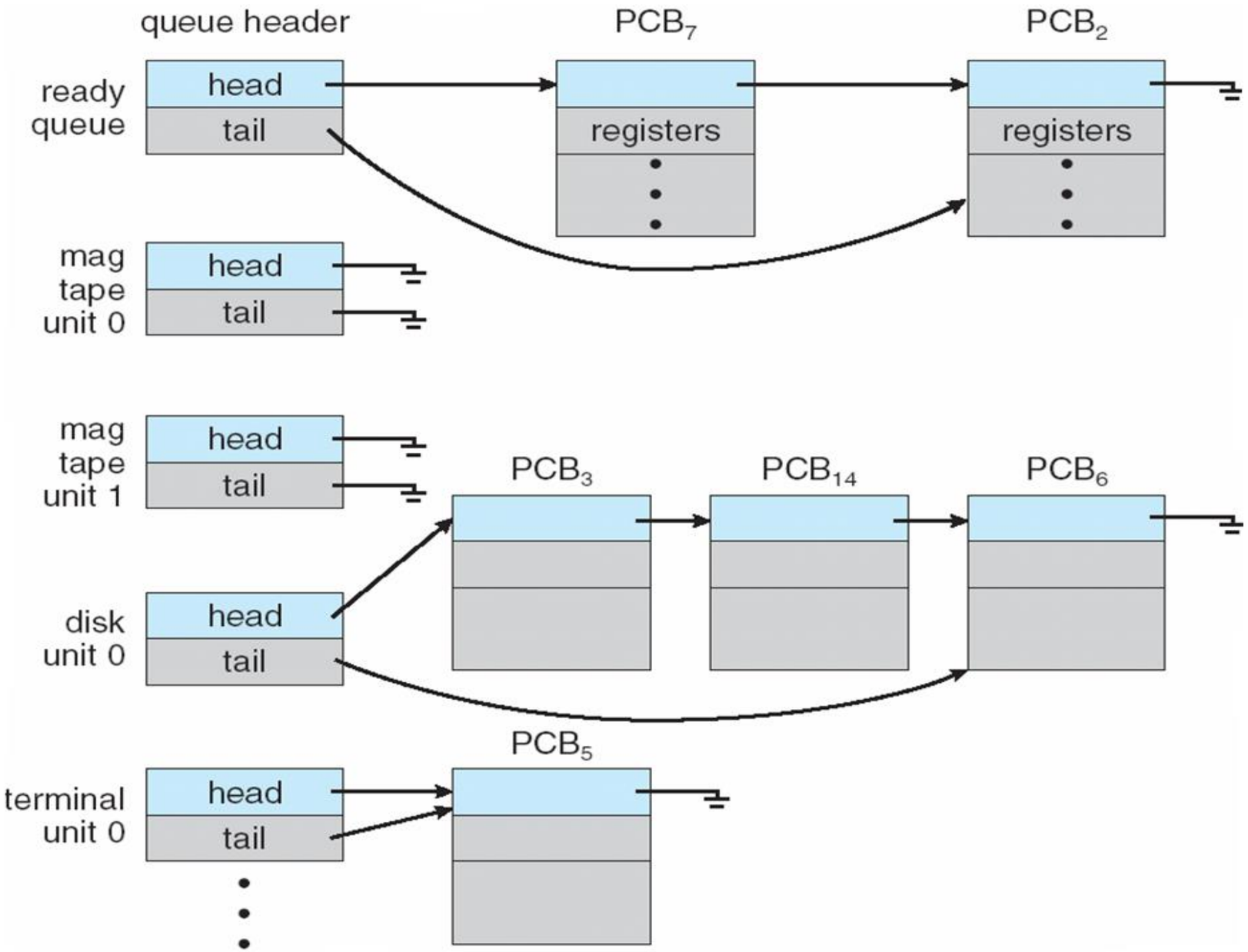
2.2 Process Scheduling

- Maximize CPU use, quickly switch processes onto CPU for time sharing
- **Process scheduler** selects among available processes for next execution on CPU
  - Scheduling Queues
  - Schedulers
  - Context Switch

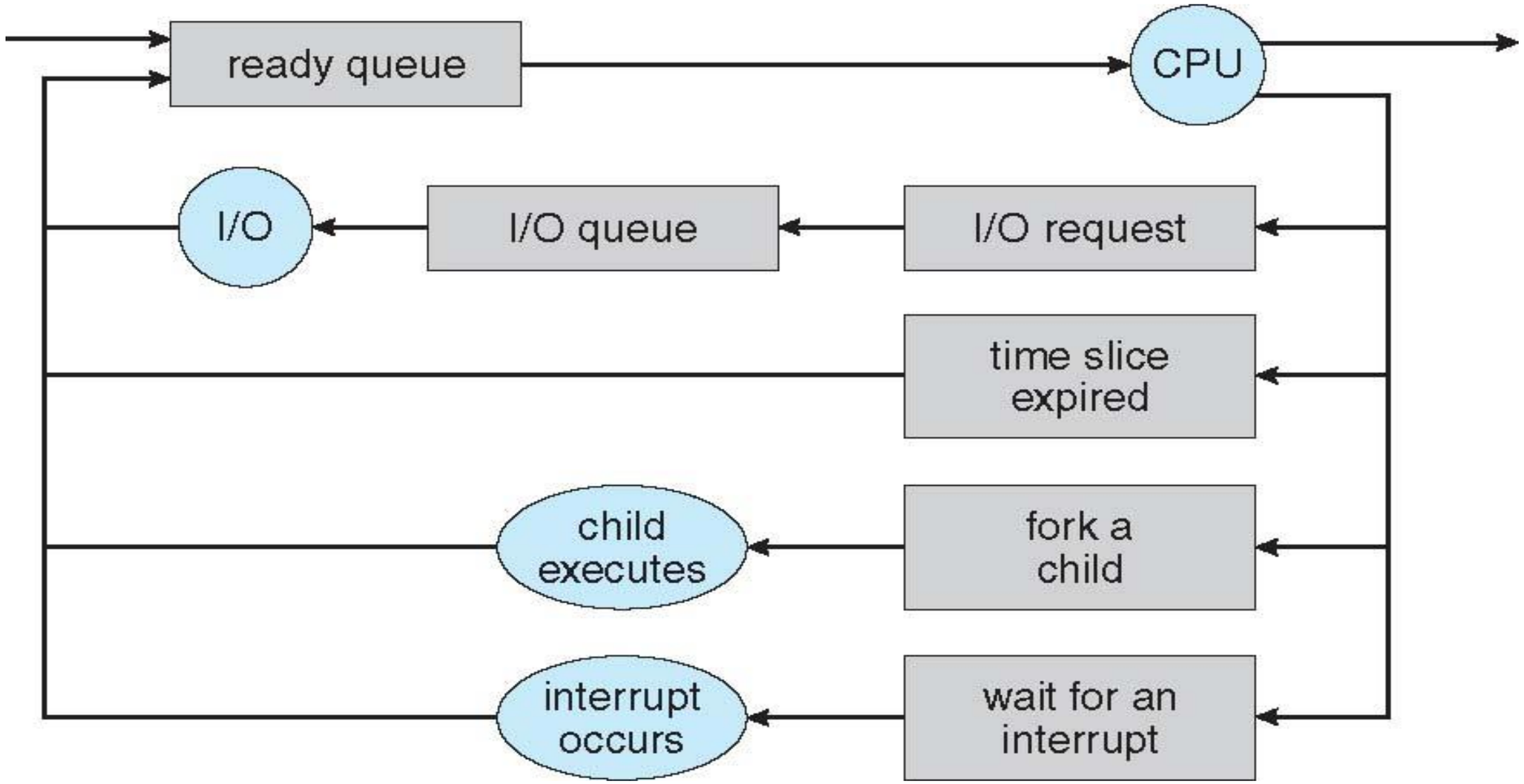
Process Scheduling – Scheduling Queues

- Process 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
- Queue is generally stored as linked list

Ready Queue And Various I/O Device Queues



Representation of Process Scheduling



Process Scheduling- Schedulers

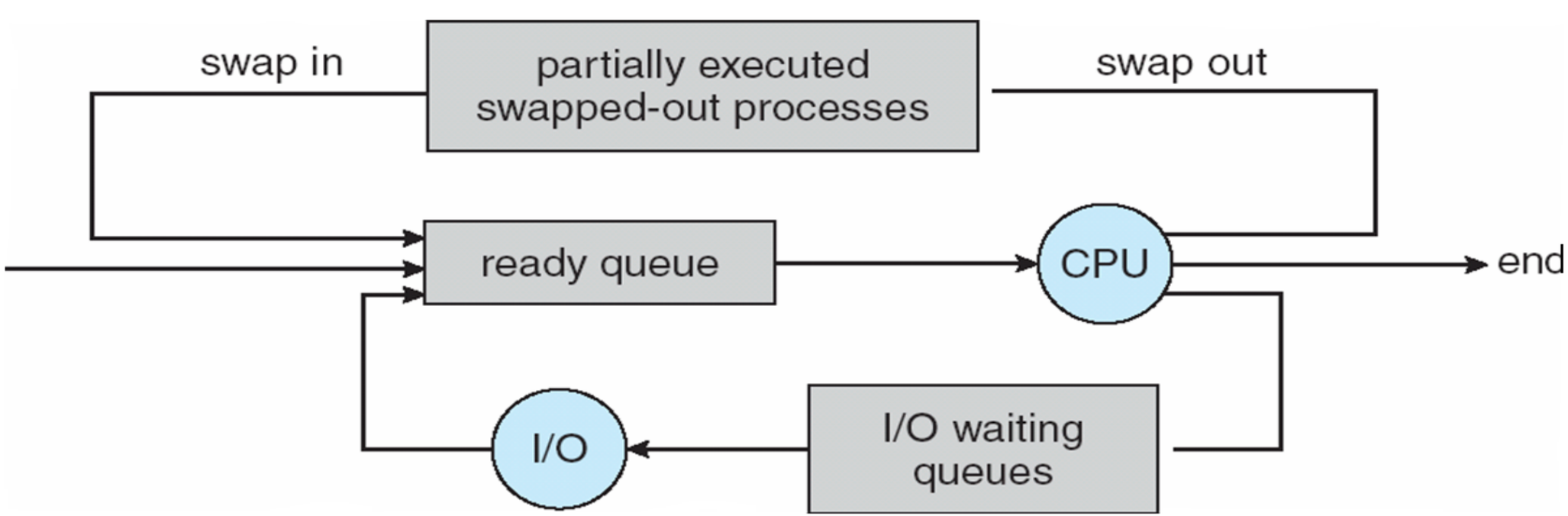
- **Long-term scheduler** (or job scheduler) – selects which processes should be brought into the ready queue
- **Short-term scheduler** (or CPU scheduler) – selects which process should be executed next and allocates CPU
  - Sometimes the only scheduler in a system
- The Schedulers will implement scheduling policies.

Schedulers (Cont.)

- **Short-term scheduler** is invoked very frequently (milliseconds) ⇒ (must be fast)
- **Long-term scheduler** is invoked very infrequently (seconds, minutes) ⇒ (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
  - **CPU-bound process** – spends more time doing computations; few very long CPU bursts

Addition of Medium Term Scheduling

- **Medium-term scheduler** (or job scheduler) – removes process from memory so that reduce the degree of multiprogramming. Later, it reintroduces in main memory. i.e, it takes care of swapping process

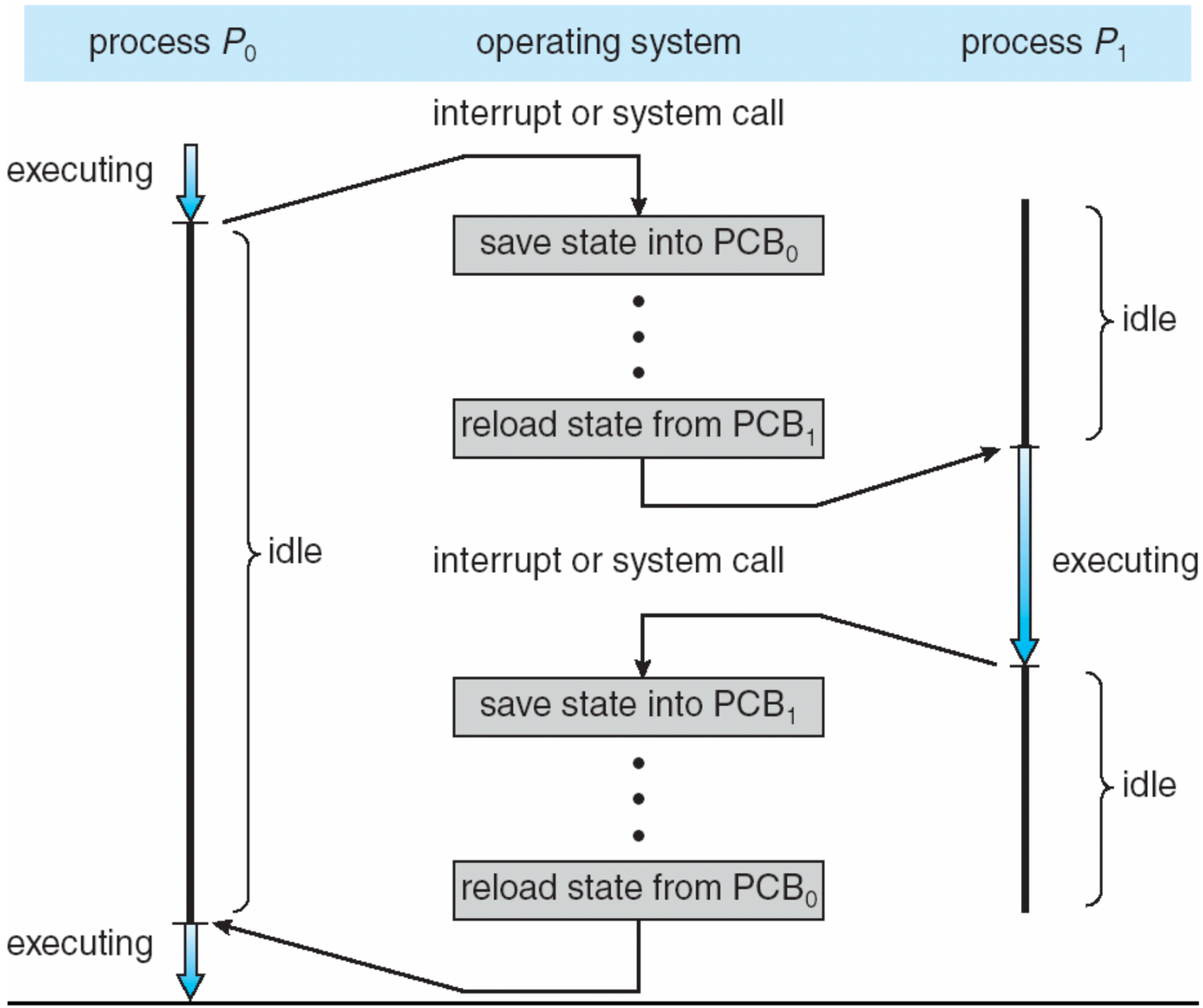


Process Scheduling - Context Switch

- 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 no useful work while switching
  - The more complex the OS and the PCB -> longer the context switch


process P_0
operating system
process P_1
interrupt or system call
executing
save state into PCB_0
reload state from PCB_1
idle
idle
interrupt or system call
executing
save state into PCB_1
reload state from PCB_0
idle
executing

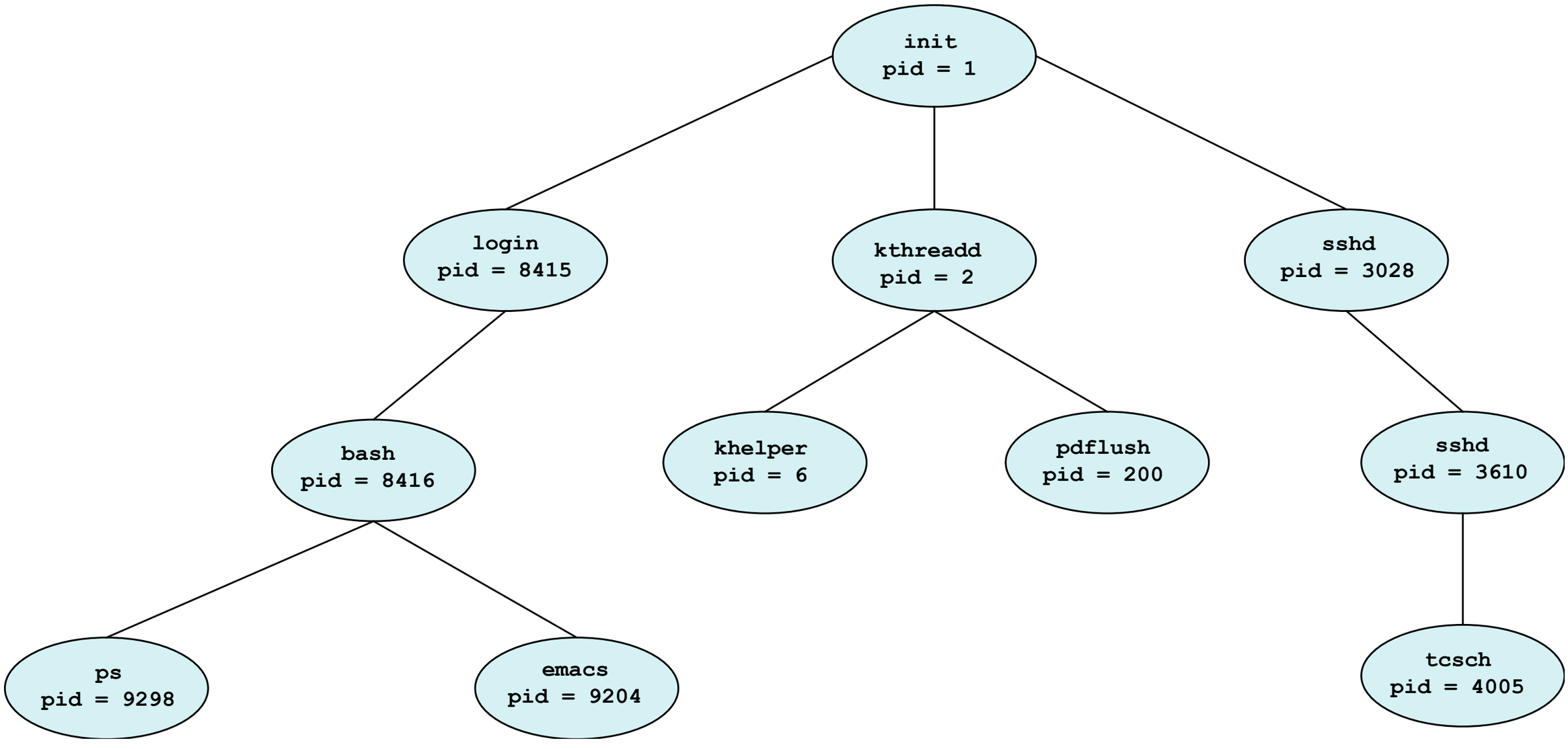
2.3 Operations on Processes

- A process in most of the systems can execute concurrently, and they must be created & deleted dynamically. Thus, these Systems must provide mechanism for
  - Process Creation
  - Process Deletion

Operations on Processes – Process Creation

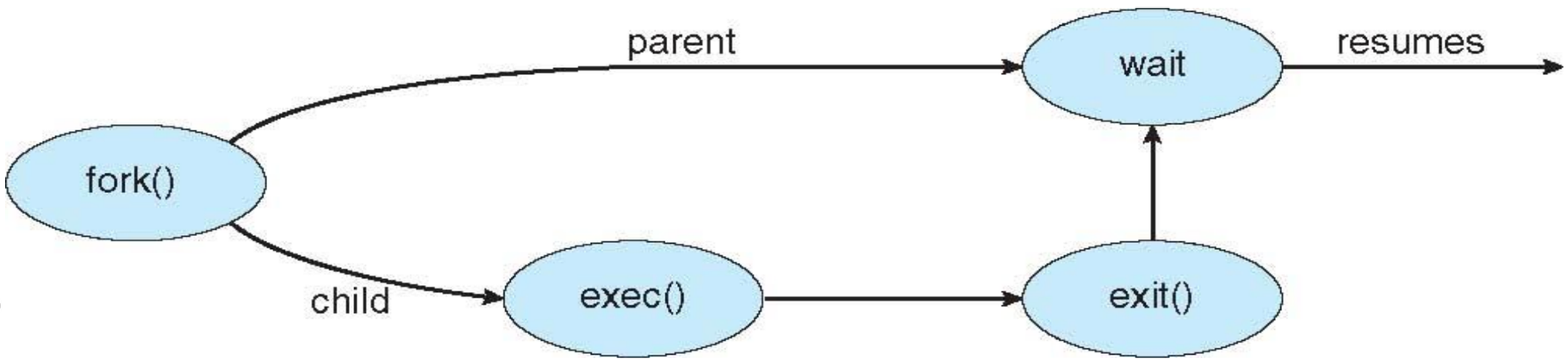
- A **(parent)** process can create several (**children)** processes
  - Children can, in turn, create other processes
  - Hence, a **tree** of processes forms
- Generally, process identified and managed via a **process identifier** (**pid**)
- Resource sharing options (of process creation)
  - Parent and children share all resources
  - Children share subset of parent's resources
    - One usage is to prevent system overload by too many child processes
  - Parent and child share no resources
- Execution options
  - Parent and children execute concurrently
  - Parent waits until children terminate

Operations on Processes – Process Creation



Operations on Processes – Process Creation

- Address space options
  - Child duplicate of parent
  - Child has a program loaded into it
- UNIX examples
  - **`fork()`** system call creates new process
    - Child is a copy of parent's address space
      - except fork() returns 0 to child and nonzero to parent
  - **`exec()`** system call used after a **`fork()`** to replace the process' memory space with a new program



D

Operations on Processes – Process 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 using the **`abort()`** system call. Some reasons for doing so:
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - The parent is exiting and the operating systems does not allow a child to continue if its parent terminates

Operations on Processes – Process Termination

- Some OS's don't allow child to exists if its parent has terminated
  - **cascading termination -** if a process terminates, then all its children, grandchildren, etc must also be terminated.
  - The termination is initiated by the operating system
- The parent process may wait for termination of a child process by using the **`wait()`** system call **.**
  - The call returns status information and the pid of the terminated process

```
pid = wait(&status);
```

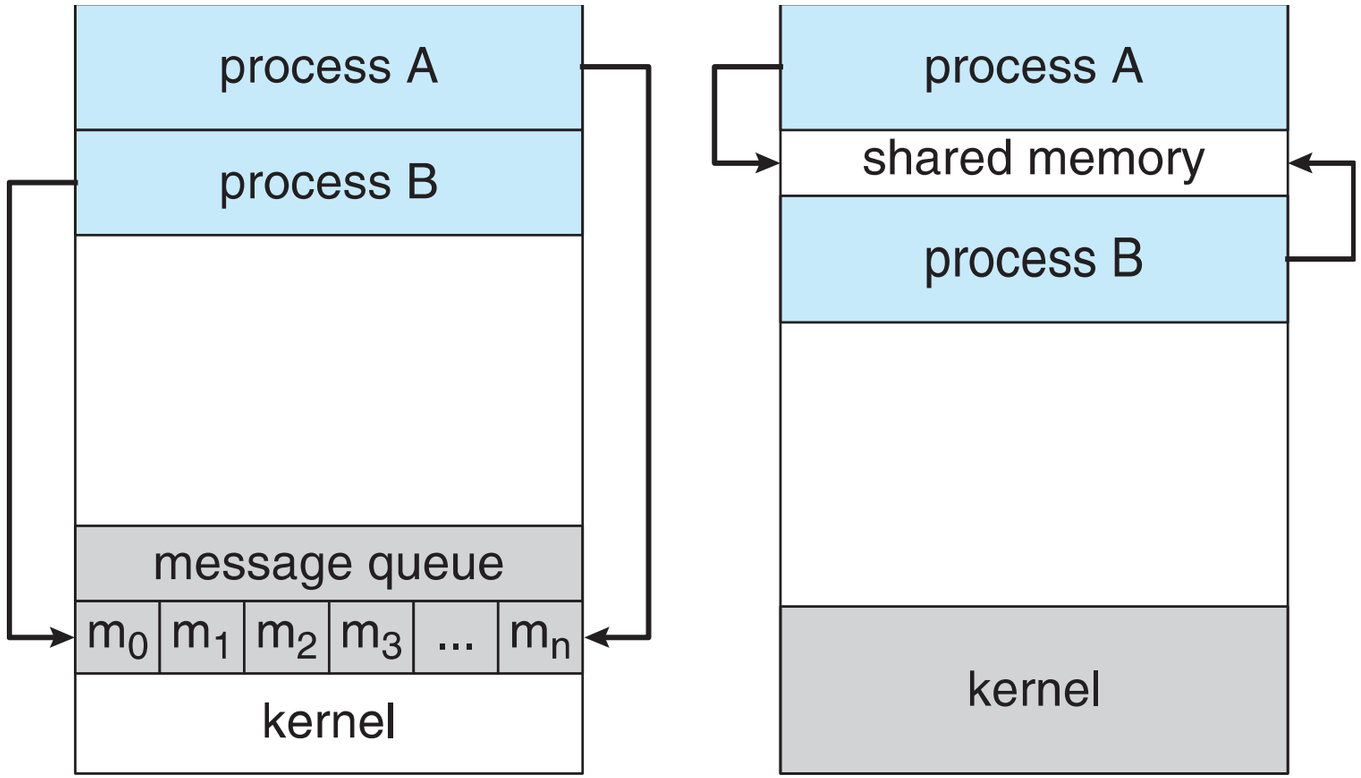
- Process finished its execution but the state not changed is a **zombie**
- If parent terminated without invoking **`wait`** , process is an **orphan**

2.4 Inter Process Communication

- Processes within a system may be ***independent*** or ***cooperating***
  - When processes execute they produce some **computational results**
  - ***Independent*** process cannot affect (or be affected) by results of another process
  - ***Cooperating*** process can affect (or be affected) by results of another process
- Advantages of process cooperation
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience

-contd..

- For fast exchange of information, cooperating processes need some **interprocess communication** (**IPC**) **mechanisms**
- Two models of IPC
  - **Shared memory**
  - **Message passing**



IPC – Shared Memory Systems

- An area of memory is shared among the processes that wish to communicate
- The communication is under the control of the users processes, not the OS.
- Major issue is to provide mechanism that will allow the user processes to synchronize their actions when they access shared memory.

IPC – Shared Memory Systems

- **Producer-consumer problem** – a common paradigm for cooperating processes
- ***Producer*** process : produces some information
- ***Consumer*** process : consumes this information
- Challenge:
  - Producer and consumer should run concurrently and efficiently
  - Producer and consumer must be synchronized
    - Consumer cannot consume an item before it is produced
- Shared-memory is solution to producer-consumer
- Uses a buffer in shared memory to exchange information
- **unbounded-buffer** assumes no practical limit on the buffer size
- **bounded-buffer** assumes a fixed buffer size

IPC – Message Passing

- Mechanism for processes to communicate and to synchronize their actions
- Message system – processes communicate with each other without resorting to shared variables
- IPC facility provides two operations:
  - send(message)
  - receive(message)
- If P and Q wish to communicate, they need to:
  - Establish a communication link between them
  - Exchange messages via send/receive

IPC – Message Passing

- Implementation of Communication link
  - Physical link (eg: Shared memory, hardware bus)
  - Logical link(eg: direct or indirect, synchronous or asynchronous)
- The message size is either fixed or variable
- Generally variable sized messages are easy to implement
- Implementation issues:
  - How are links established?
  - Can a link be associated with more than two processes?
  - How many links can there be between every pair of communicating processes?
  - What is the capacity (buffer size) of a link?
  - Is the size of a message that the link can accommodate fixed or variable?
  - Is a link unidirectional or bi-directional?

IPC MessagePassing – Naming- Direct Communication

- Processes must name each other explicitly:
  - **`send`** (*P, message*) – send a message to process P
  - **`receive`**(*Q, message*) – receive a message from process Q

- Properties of a direct communication link
  - Links are established automatically
  - A link is associated with exactly one pair of communicating processes
  - Between each pair there exists exactly one link
  - The link is usually bi-directional

IPC MessagePassing- Naming – Indirect Communication

- Messages are directed and received from mailboxes (also referred to as ports)
  - Each mailbox has a unique id
  - Processes can communicate only if they share a mailbox

- Properties of an indirect communication link
  - Link established only if both members of the pair have a shared mailbox
  - A link may be associated with many processes
  - Each pair of processes may share several communication links
  - Link may be unidirectional or bi-directional

IPC – Message Passing- Indirect Communication

Operations

- create a new mailbox (port)
- send and receive messages through mailbox
- destroy a mailbox

Primitives are defined as:

send(*A*, *message*) – send a message to mailbox A

receive(*A*, *message*) – receive a message from mailbox A

IPC – Message Passing - Synchronization

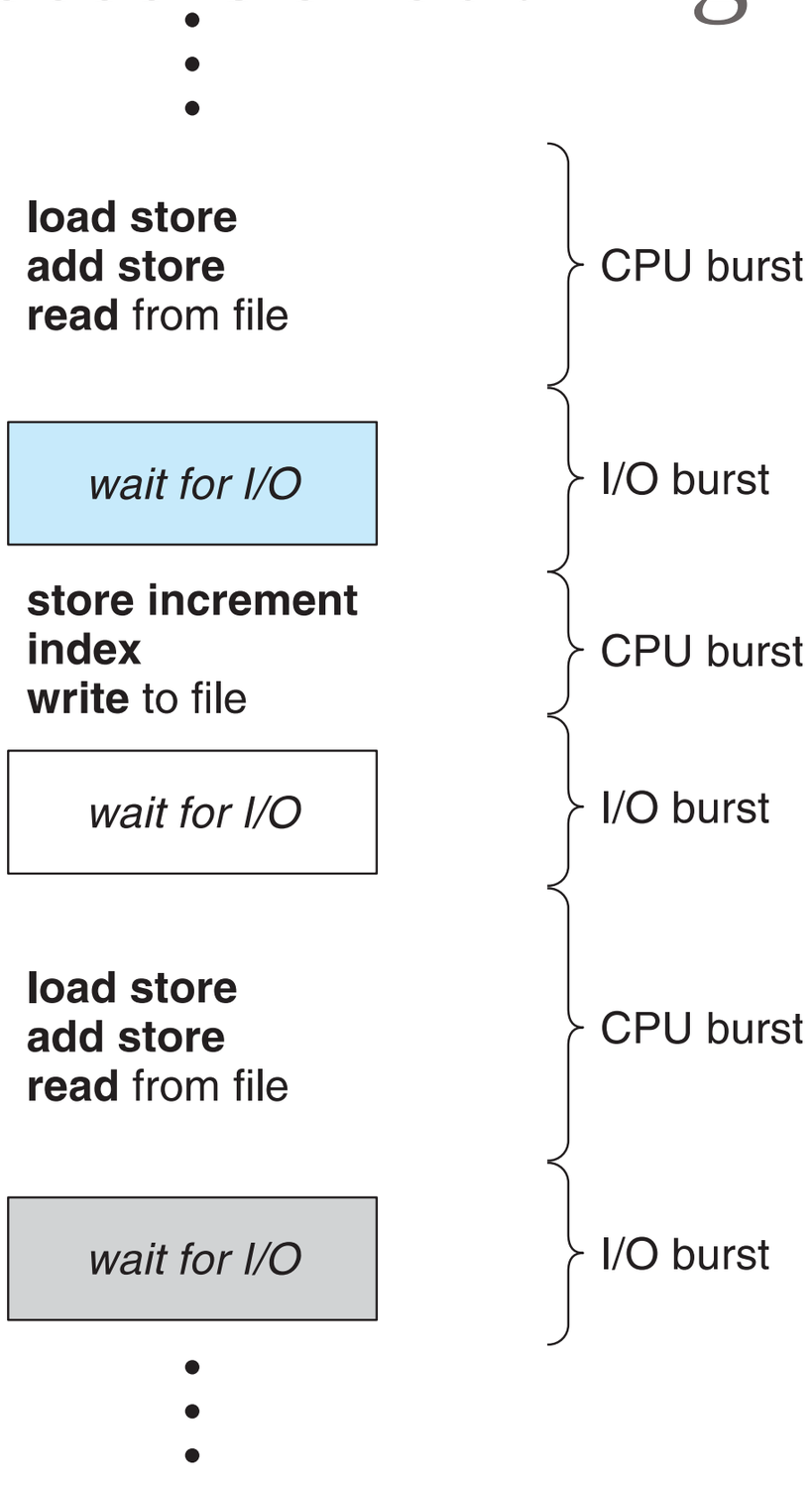
- Message passing may be either
  - Blocking, or
  - Non-blocking
- **Blocking** is considered **synchronous**
  - **Blocking send** -- the sender is blocked until the message is received
  - **Blocking receive** -- the receiver is blocked until a message is available
- **Non-blocking** is considered **asynchronous**
  - **Non-blocking send** -- the sender sends the message and continues sending
  - **Non-blocking receive** -- the receiver receives:
    - A valid message, or
    - Null message
- Different combinations possible
  - If both send and receive are blocking – called a **rendezvous**

IPC – Message Passing - Buffering

- Queue of messages is attached to the link.
- Implemented in one of three ways
  1. Zero capacity – no messages are queued on a link.
     - Sender must wait for receiver
  2. Bounded capacity – finite length of *n* messages
     - Sender must wait if link full
  3. Unbounded capacity – infinite length
     - Sender never waits

Basics Concepts in Process Scheduling

- Maximum CPU utilization obtained with multiprogramming
  - waiting for I/O is wasteful
  - 1 thread will utilize only 1 core
- CPU–I/O Burst Cycle
  - Process execution consists of:
    - a **cycle** of CPU execution
    - and I/O wait
- **CPU burst** followed by **I/O burst**
- CPU burst distribution is of main concern



Basics Concepts in Process Scheduling

CPU Scheduler

- **Short-term scheduler**
  - Selects 1 process from the ready queue
    - then allocates the CPU to it
  - Queue may be ordered in various ways : FIFO Queue, Priority Queue, tree, linked list
- 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 called **nonpreemptive (=cooperative)**
- All other scheduling is called **preemptive**
  - Process can be interrupted and must release the CPU
  - Special care should be taken to prevent problems that can arise
    - Access to shared data – race condition can happen, if not handled

Basics Concepts in Process Scheduling

Dispatcher

- **Dispatcher**
  - a module that 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
  - This time should be as small as possible

Scheduling Criteria

- How do we decide which scheduling algorithm is good?
- Many criteria for judging this has been suggested
  - Which characteristics considered can change significantly which algorithm is considered the best
- **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 to start responding
  - Used for interactive systems
  - Time from when a request was submitted until the first response is produced

Scheduling Criteria

Scheduling Algorithm Optimization Criteria

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

Scheduling Algorithm

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: $(0 + 24 + 27)/3 = 17$

Scheduling Algorithm 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
  - Hence, average waiting time of FCFS not minimal
  - And it may vary substantially
- FCFS is nonpreemptive
  - Not a good idea for timesharing systems
- FCFS suffers from the convoy effect, shorter job to wait for a longer time

Scheduling Algorithm Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst
  - SJF uses these lengths to schedule the process with the shortest time
- Notice, the burst is used by SJF,
  - **not** the process end-to-end running time
    - implied by word "job" in SJF
  - Hence, it should be called ``Shorted-Next-CPU-Burst"
  - However, "job" is used for historic reasons
- Two versions of SJF: preemptive and nonpreemptive
  - Assume
    - A new process Pnew arrives while the current one Pcur is still executing
    - The burst of Pnew is less than what is left of Pcur
  - Nonpreemptive SJF – will let Pcur finish
  - Preemptive SJF – will preempt Pcur and let Pnew execute
    - This is also called shortest-remaining-time-first scheduling

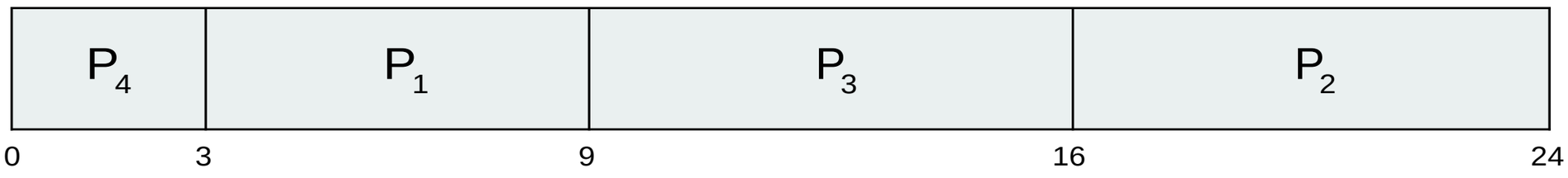
Scheduling Algorithm SJF (Cont.)

- **Advantage:**
  - SJF is **optimal** in terms of the average waiting time
- **Challenge of SJF:**
  - Hinges on knowing the length of the next CPU burst
    - But how can we know it?
    - Solutions: ask user or estimate it
  - In a batch system and long-term scheduler
    - Could ask the user for the job time limit
    - The user is motivated to accurately estimate it
      - Lower value means faster response
      - Too low a value will cause time-limit violation and job rescheduling
  - In a short-term scheduling
    - Use estimation
    - Will be explained shortly

Scheduling Algorithm 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

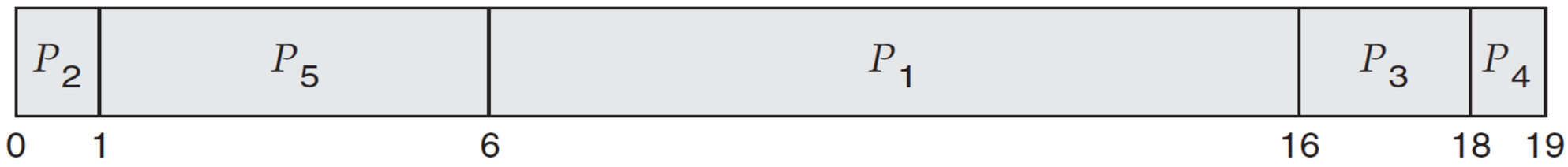
Scheduling Algorithm Priority Scheduling

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

Scheduling Algorithm

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Priority scheduling Gantt Chart



- Average waiting time = 8.2 msec

Scheduling Algorithm Round Robin (RR)

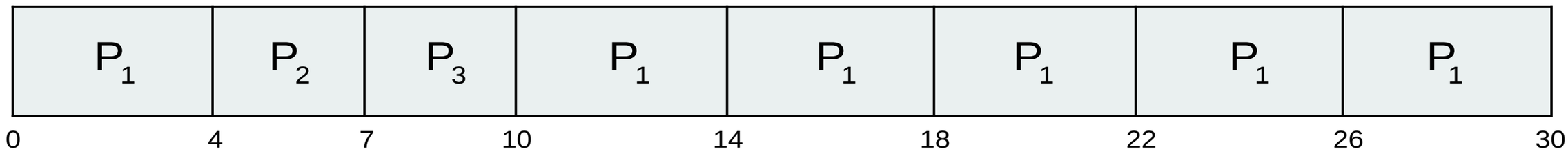
- Each process gets a small unit of CPU time
  - **Time quantum** q
  - Usually 10-100 milliseconds
- After this time has elapsed:
  - the process is preempted and
  - added to the end of the ready queue
- The process might run for $\leq q$ time
  - For example, when it does I/O
- **If**
  - n processes in the ready queue, and
  - the time quantum is q
- **then**
  - "Each process gets $1/n$ of the CPU time"
    - Incorrect statement from the textbook
  - in chunks of $\leq q$ time units at once
  - Each process waits $\leq (n\text{-}1)q$ time units

Scheduling Algorithm

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:



- Typically:
  - Higher average turnaround (end-to-end running time) than SJF
  - But better ***response*** than SJF

Scheduling Algorithm - Multilevel Queue

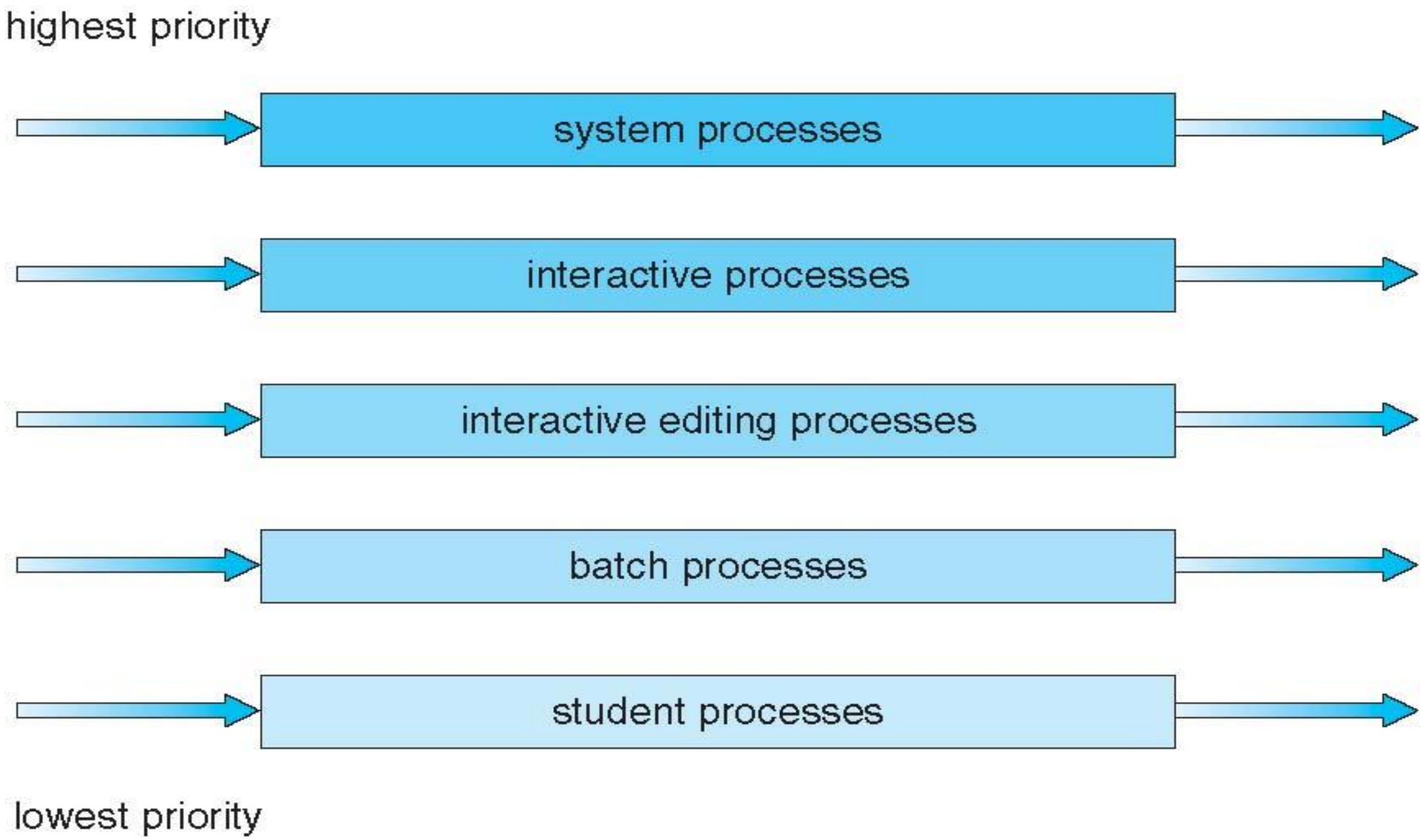
- Another class of scheduling algorithms when processes are classified into groups, for example:
  - **foreground** (interactive) processes
  - **background** (batch) processes
- Ready queue is partitioned into separate queues, e.g.:
  - **Foreground** and **background** queues
- Process is permanently assigned to one queue
- Each queue has its own scheduling algorithm, e.g.:
  - foreground – RR
  - background – FCFS

Scheduling Algorithm - Multilevel Queue

Scheduling must be done between the queues:

- Fixed priority scheduling
  - For example, foreground queue might have absolute priority over background queue
    - Serve all from foreground then from background
    - May lead to starvation
- Time slice scheduling
  - Each queue gets a certain amount of CPU time which it can schedule amongst its processes, e.g.:
    - 80% to foreground in RR
    - 20% to background in FCFS

Scheduling Algorithm - Multilevel Queue



Scheduling Algorithm - Multilevel Feedback Queue

- The previous setup: a process is permanently assigned to one queue
  - **Advantage:** Low scheduling overhead
  - **Disadvantage:** Inflexible

- Multilevel-feedback-queue scheduling algorithm
  - Allows a process to move between the various queues
    - More flexible
  - **Idea:** separate processes based on the characteristics of their CPU bursts
  - If a process uses too much CPU time => moved to lower-priority queue
    - Keeps I/O-bound and interactive processes in the high-priority queue
  - A process that waits too long can be moved to a higher priority queue
    - This form of aging can prevent starvation

Scheduling Algorithm - Multilevel Feedback Queue

- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service
- Multilevel-feedback-queue scheduler
  - The most general CPU-scheduling algorithm
  - It can be configured to match a specific system under design
  - Unfortunately, it is also the most complex algorithm
    - Some means are needed to select values for all the parameters

Scheduling Evaluation

- How to select CPU-scheduling algorithm for an OS?
  - Determine criteria, then evaluate algorithms
- Evaluation Methods
  - Deterministic modeling
  - Queuing models
  - Simulations
  - Implementation