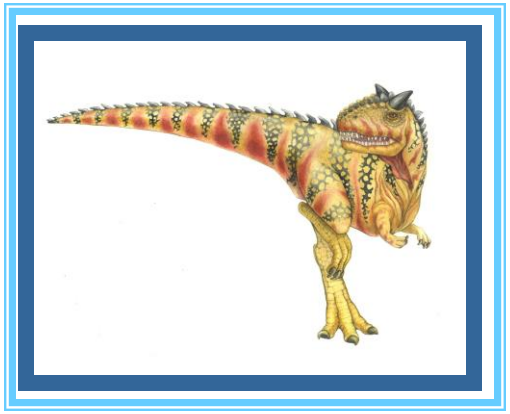
UNIT-2
Processes Management



Process Concept

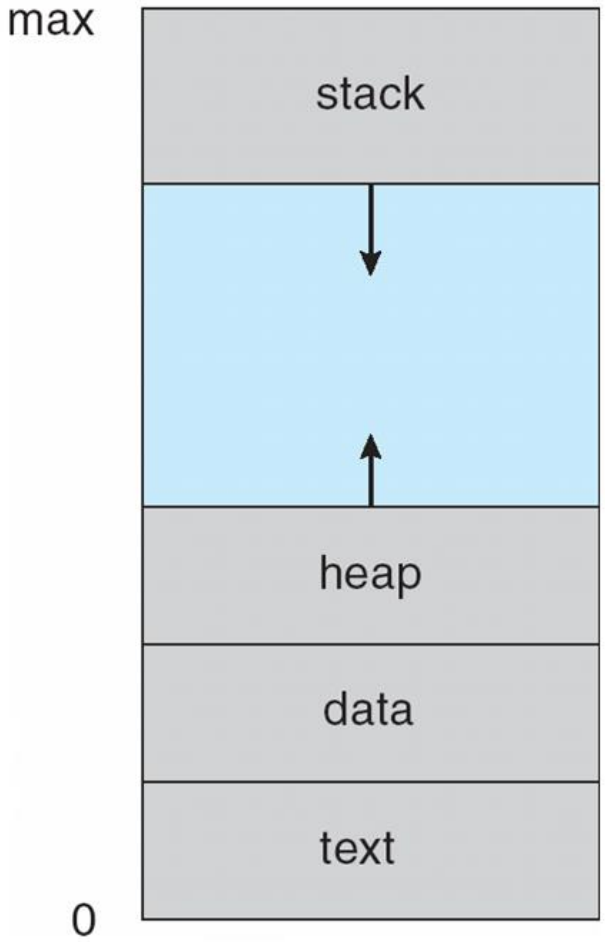
- **Definition:** A **process** is a program in execution.
- **Process** – a program in execution; process execution must progress in sequential fashion
- An operating system executes a variety of programs:
  - Batch system – jobs
  - Time-shared systems – user programs or tasks
- Textbook uses the terms *job* and *process* almost interchangeably
- A process includes:
  - program counter
  - stack
  - data section

Process in Memory

- ✓A process is more than the program code, which is sometimes known as the **text section**.
- ✓It also includes the current activity, as represented by the value of the program counter and the contents of the processor's registers.
- ✓A process generally includes the process stack, which contains temporary data (such as function parameters, return addresses, and local variables), and a data section, which contains global variables.
- ✓A process may also include a heap, which is memory that is dynamically allocated during process run time.

Process in Memory

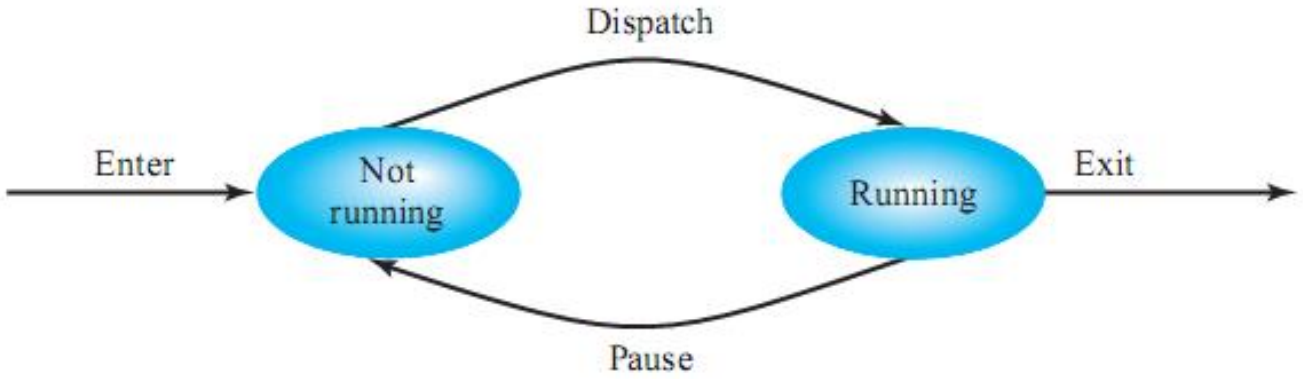
The structure of a process in memory is shown in Figure 3.1.



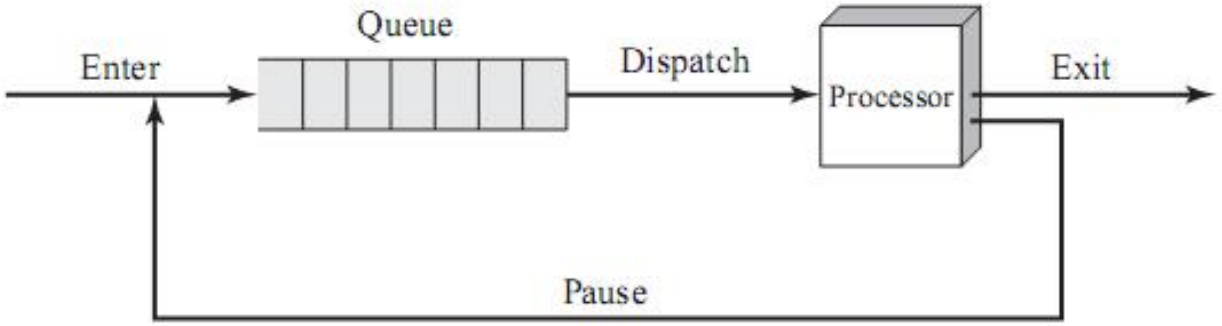
Process State

- As a process executes, it changes state.
- The state of a process is defined in part by the current activity of that process.
- Each process may be in one of the following states:
  - ✓ **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

A Two-State Process Model



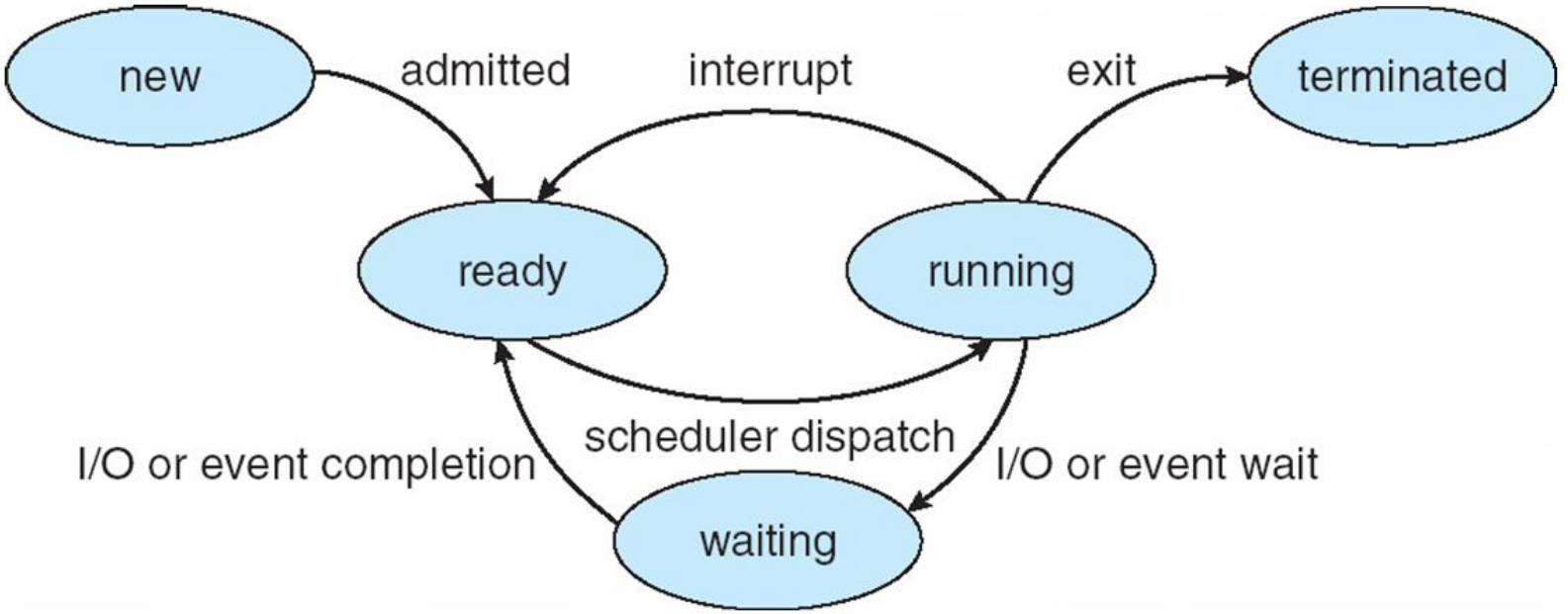
(a) State transition diagram



(b) Queuing diagram

Figure 3.5 Two-State Process Model

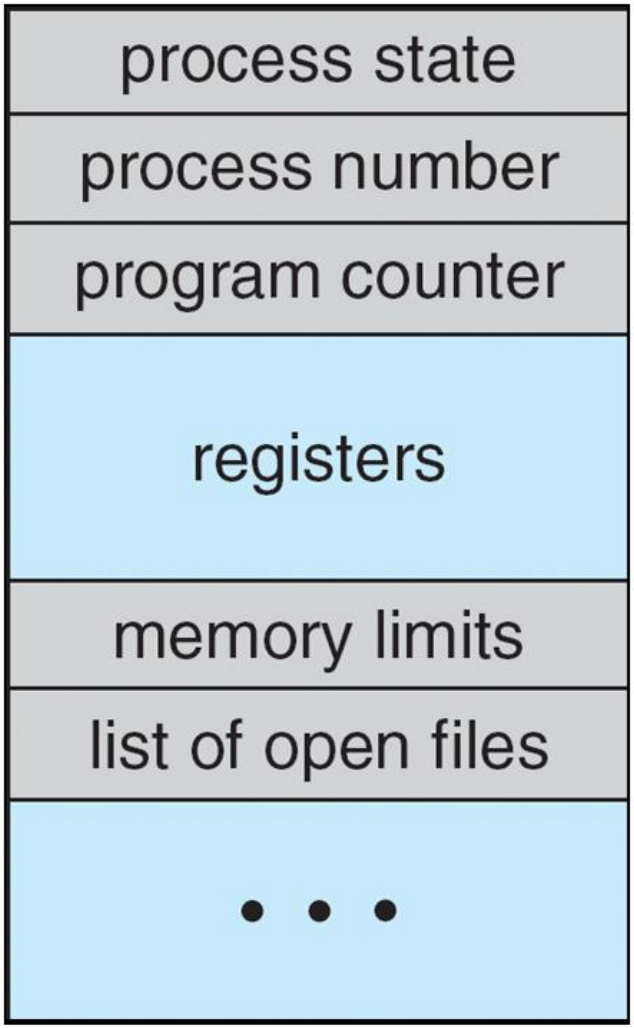
Diagram of Process State



Process Control Block (PCB)

- A process in an operating system is represented by a data structure known as a process control block (PCB) or process descriptor.
- The PCB contains important information about the specific process including
  - Process state
  - Program counter
  - CPU registers
  - CPU scheduling information
  - Memory-management information
  - Accounting information
  - I/O status information

Process Control Block (PCB)



Process Control Block (PCB)

Process state. The state may be new, ready running, waiting, halted, and so on.

Program counter. The counter indicates the address of the next instruction to be executed for this process.

CPU registers. The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code information.

Process Control Block (PCB)

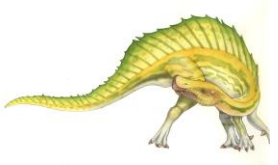
CPU-scheduling information. This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

Memory-management information. This information may include such information as the value of the base and limit registers, the page tables, or the segment tables, depending on the memory system used by the operating system.

Accounting information. This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.

I/O status information. This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

Process Scheduling



Process Scheduling

Definition

✓The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

✓Process scheduling is an essential part of a Multiprogramming operating system.

✓Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

Scheduling Queues

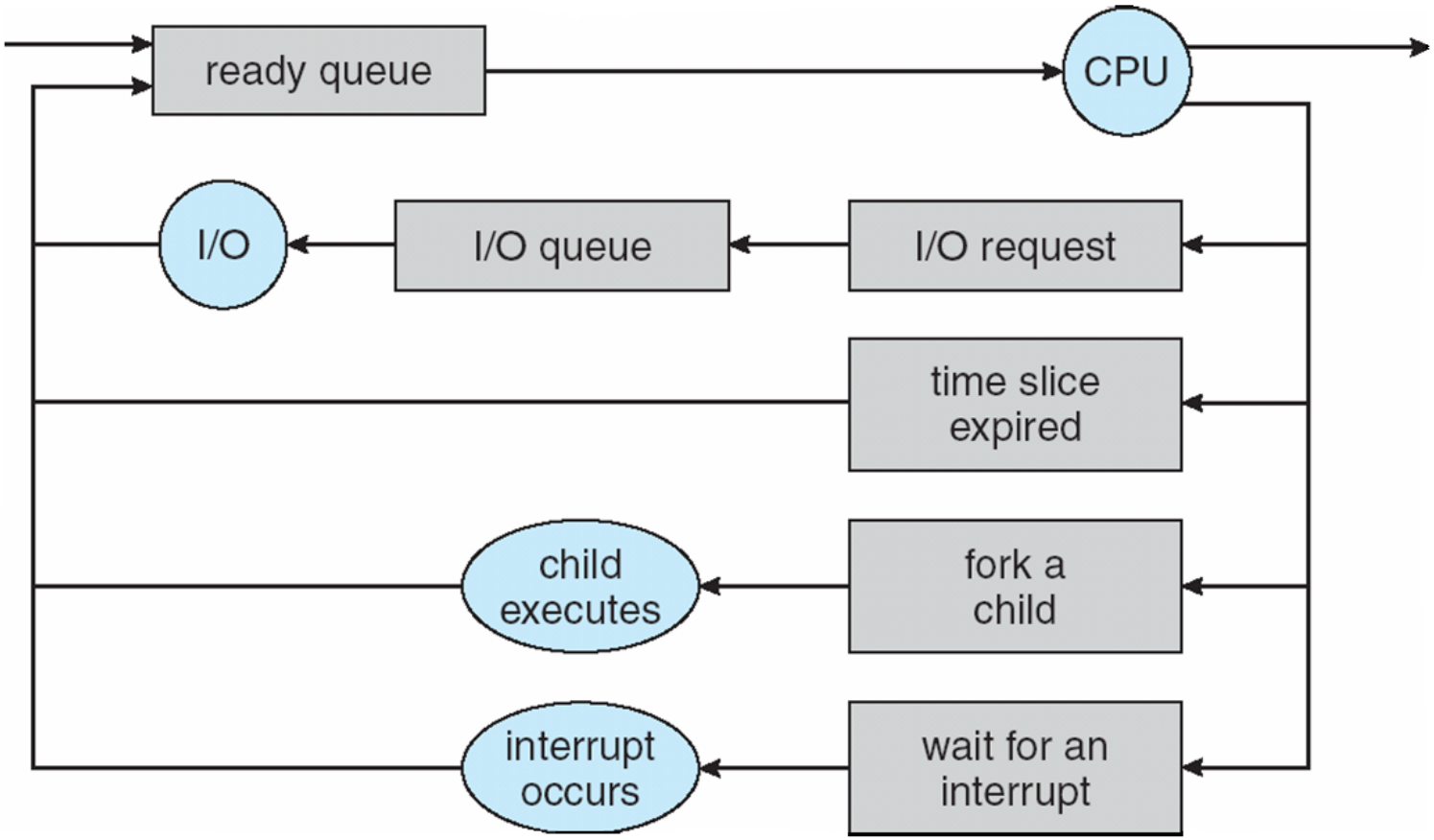
Scheduling queues refers to queues of processes or devices.

Job queue – set of all processes in the system.

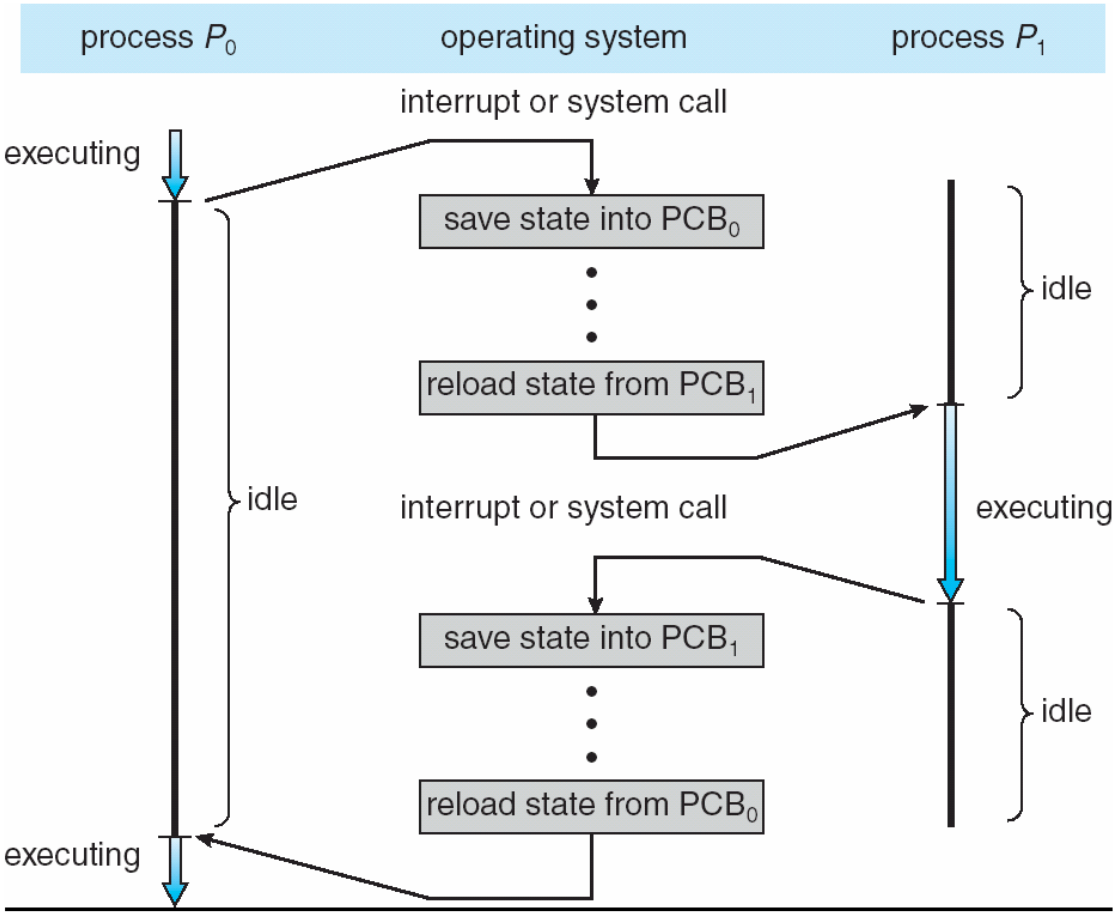
Ready queue – set of all processes residing in main memory, ready and waiting to execute.

Device queues – set of processes waiting for an I/O device
Processes migrate among the various queues

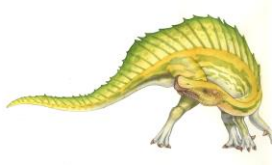
Representation of Process Scheduling



CPU Switch From Process to Process



Schedulers



Schedulers

✓Schedulers are special system software's which handles process scheduling in various ways.

✓Their main task is to select the jobs to be submitted into the system and to decide which process to run.

✓Schedulers are of three types

- **Long Term Scheduler**
- **Short Term Scheduler**
- **Medium Term Scheduler**

Long Term Scheduler:

- It is also called job scheduler.
- Long term scheduler determines which programs are admitted to the system for processing.
- Job scheduler selects processes from the queue and loads them into memory for execution.
- Process loads into the memory for CPU scheduling.
- On some systems, the long term scheduler may not be available or minimal.
- Time-sharing operating systems have no long term scheduler.
- When process changes the state from new to ready, then there is use of long term scheduler..

Short Term Scheduler

- It is also called CPU scheduler.
- Main objective is increasing system performance in accordance with the chosen set of criteria.
- It is the change of ready state to running state of the process.
- CPU scheduler selects process among the processes that are ready to execute and allocates CPU to one of them.
- Short term scheduler also known as dispatcher, execute most frequently and makes the fine grained decision of which process to execute next.
- Short term scheduler is faster than long term scheduler.

Medium Term Scheduler

- ✓Medium term scheduling is part of the swapping.
- ✓It removes the processes from the memory.
- ✓It reduces the degree of multiprogramming.
- ✓The medium term scheduler is in-charge of handling the swapped out-processes.

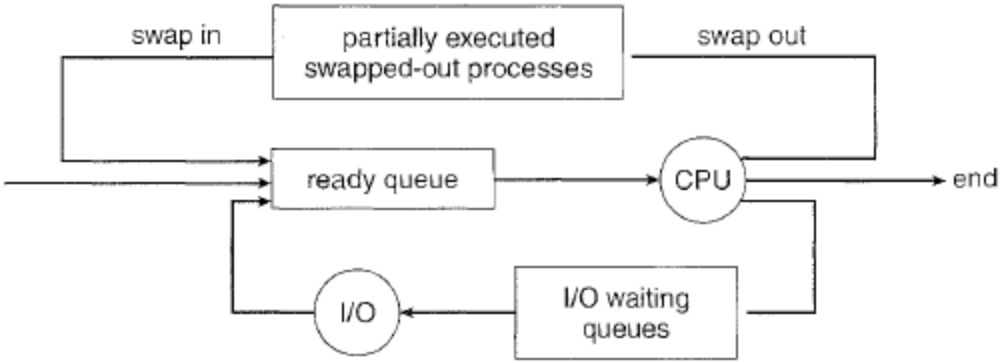
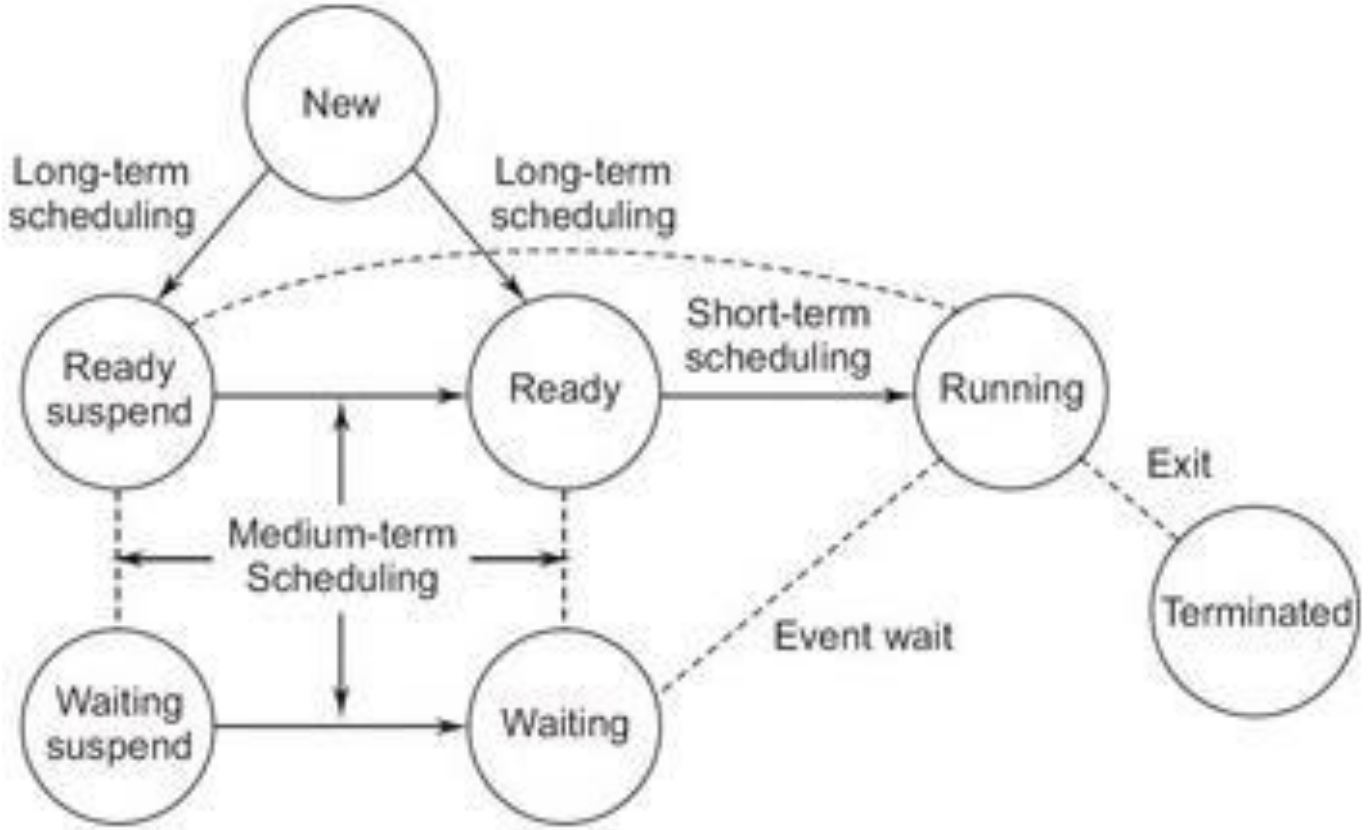
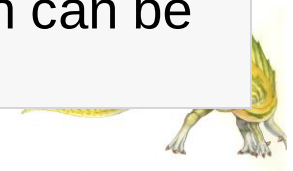


Figure 3.8 Addition of medium-term scheduling to the queueing diagram.


New
Long-term scheduling
Long-term scheduling
Ready suspend
Ready
Short-term scheduling
Running
Exit
Medium-term Scheduling
Terminated
Event wait
Waiting suspend
Waiting

Comparison between Scheduler

S. N.	Long Term Scheduler	Short Term Scheduler	Medium Term Scheduler
1	It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
2	Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short and long term scheduler.
3	It controls the degree of multiprogramming	It provides lesser control over degree of multiprogramming	It reduces the degree of multiprogramming.
4	It is almost absent or minimal in time sharing system	It is also minimal in time sharing system	It is a part of Time sharing systems.
5	It selects processes from pool and loads them into memory for execution	It selects those processes which are ready to execute	It can re-introduce the process into memory and execution can be continued.



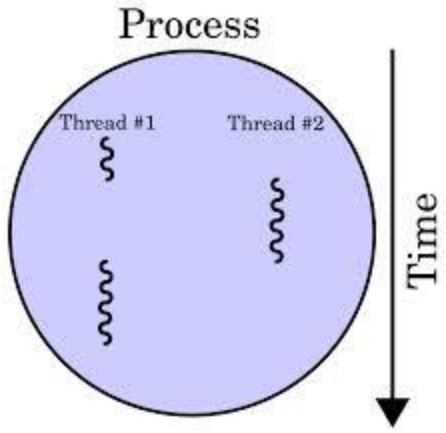
Multi Thread programming models

What is Thread?

•It is the basic unit of CPU utilization.

•A thread is a flow of execution through the process code, with its own program counter, system registers and stack.

•A thread is also called a light weight process.

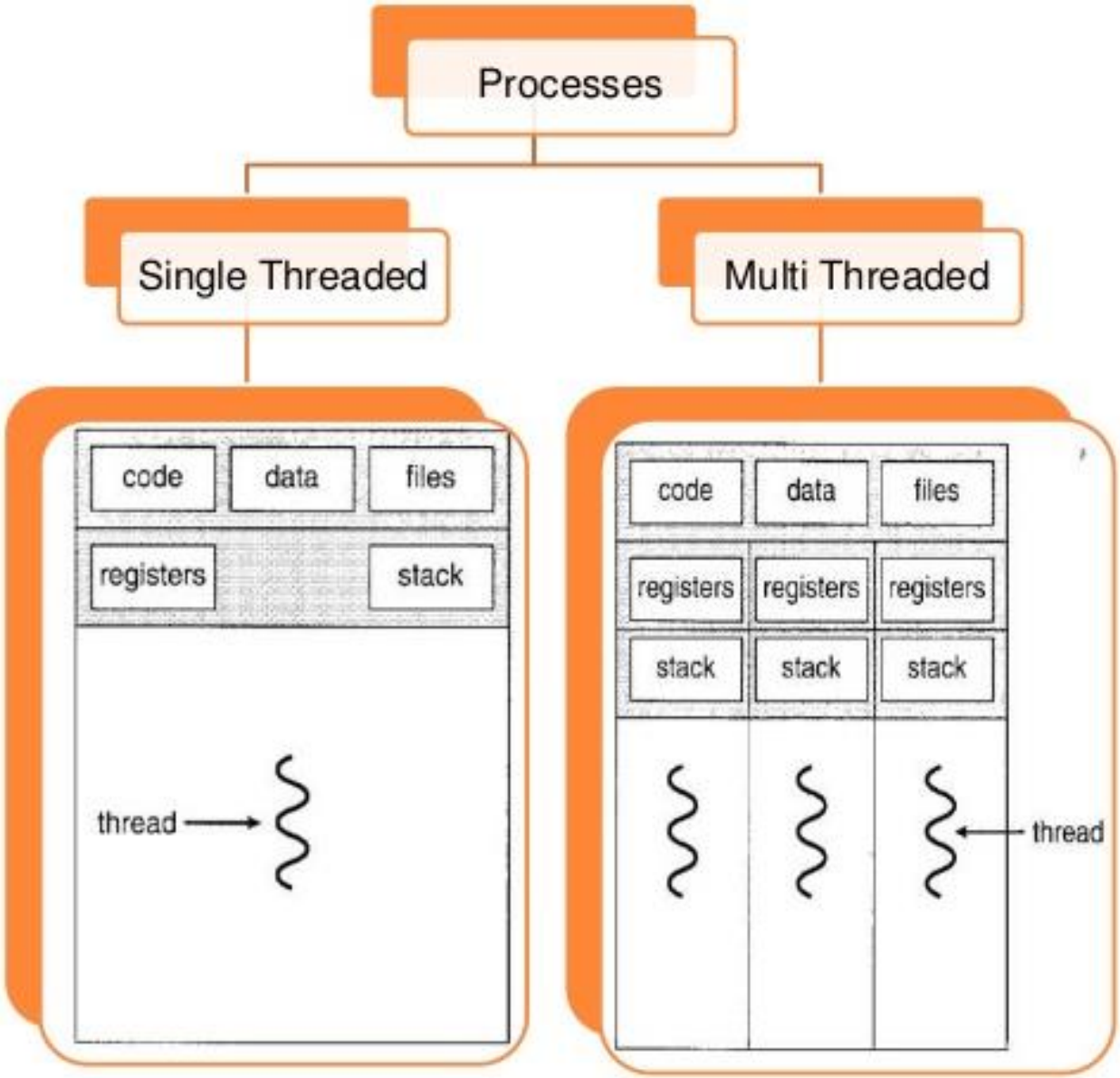


MOTIVATION

Many software packages that run on modern desktop PCs are multithreaded.

•**Example:**

•A word processor may have a thread for displaying graphics, another for responding to keystrokes, and a third is for performing spelling and grammar checking in background.



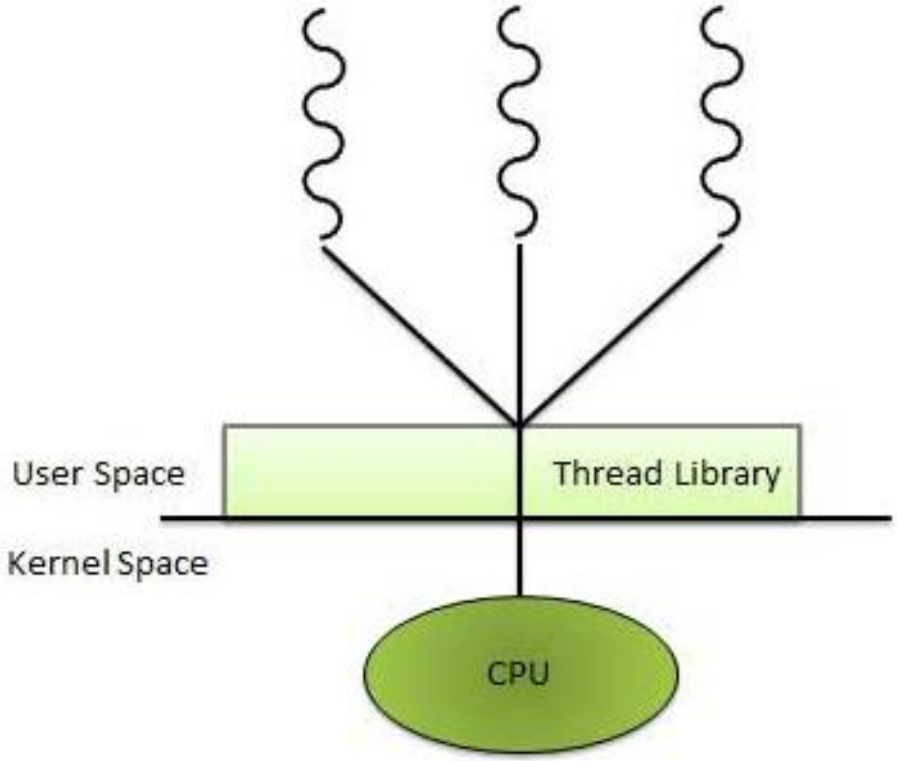
Single-threaded and multithreaded processes

Types of Thread

- **User threads** - management done by user-level threads library
- Three primary thread libraries:
  - POSIX **Pthreads**
  - Windows threads
  - Java threads
- **Kernel threads** - Supported by the Kernel
- Examples – virtually all general purpose operating systems, including:
  - Windows
  - Solaris
  - Linux
  - Tru64 UNIX
  - Mac OS X

Difference between User Level & Kernel Level Thread

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


User Space
Thread Library
Kernel Space
CPU

Multithreading Models

✓Some operating system provide a combined user level thread and Kernel level thread facility.

✓Solaris is a good example of this combined approach.

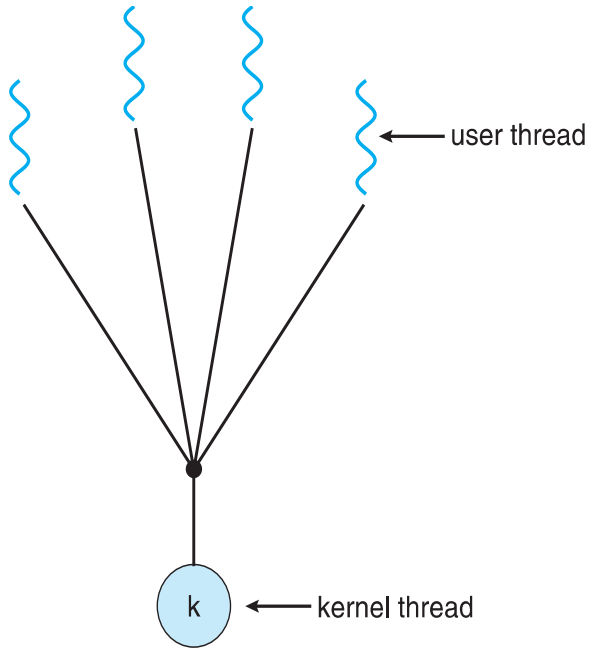
✓In a combined system, multiple threads within the same application can run in parallel on multiple processors.

✓Multithreading models are three types

- **Many to many relationship.**
- **Many to one relationship.**
- **One to one relationship.**

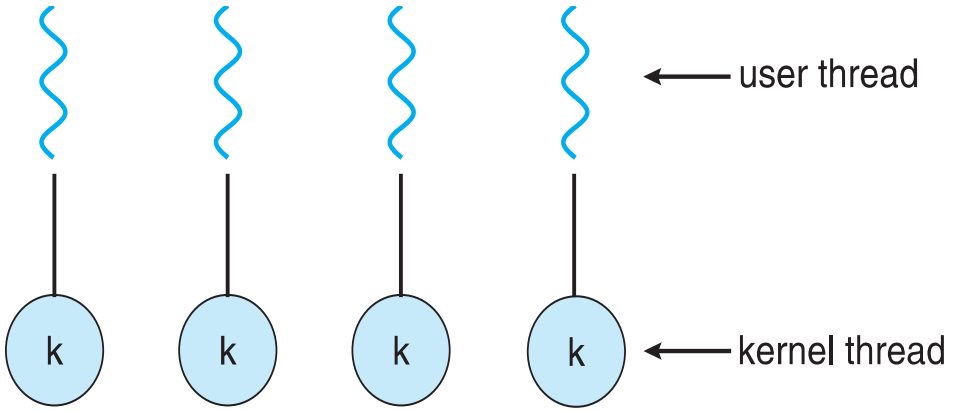
Many-to-One Model

- Many user-level threads mapped to single kernel thread
- One thread blocking causes all to block
- Multiple threads may not run in parallel on muticore system because only one may be in kernel at a time
- Few systems currently use this model
- Examples:
  - **Solaris Green Threads**
  - **GNU Portable Threads**



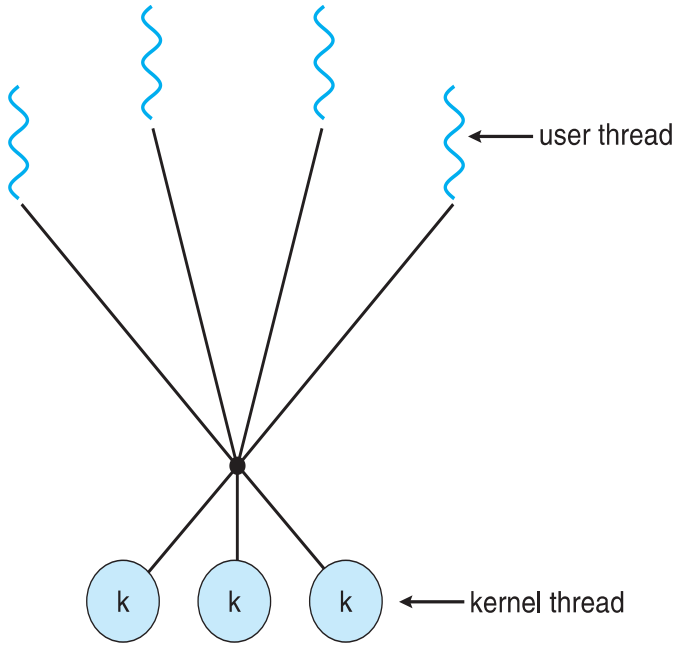
One-to-One Model

- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
- More concurrency than many-to-one
- Number of threads per process sometimes restricted due to overhead
- **Examples**
  - **Windows**
  - **Linux**
  - **Solaris 9 and later**



Many-to-Many Model

- Allows many user level threads to be mapped to many kernel threads
- Allows the operating system to create a sufficient number of kernel threads
- Solaris prior to version 9
- Windows with the *ThreadFiber* package



BENEFITS:

The benefits of multithreaded programming can be broken down into four major categories:

- **Responsiveness.**
- **Resource Sharing**
- **Economy.**
- **Utilization of multiprocessor architectures**

CPU Scheduling

Basic Concepts

✓During its lifetime, a process goes through a sequence of **CPU and I/O bursts**.

✓The ***CPU scheduler (a.k.a. short-term scheduler)*** will select one of the processes in the ready queue for execution.

✓ The CPU scheduler algorithm may have tremendous effects on the **system performance**

Non-preemptive vs. Preemptive Scheduling

- **Nonpreemptive**

  Once a process is allocated the CPU, it does not leave unless:

  - **it has to wait, e.g., for I/O request**
  - **it terminates**

- **Preemptive**

  - **OS can force (preempt) a process from CPU at anytime**
  - **E.g., to allocate CPU to another higher-priority process**

Scheduling Criteria

Several criteria can be used to compare the performance of scheduling algorithms

CPU utilization – keep the CPU as busy as possible

Throughput – 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 – Time used by a system to respond to a User Job. (seconds)

Optimization Criteria

- ✓ Maximize the CPU utilization
- ✓ Maximize the throughput
- ✓ Minimize the (average) turnaround time
- ✓ Minimize the (average) waiting time
- ✓ Minimize the (average) response time

CPU Scheduling Algorithms
Or
Process Scheduling Algorithms

Scheduling Algorithms

We'll discuss four major scheduling algorithms here which are following

- **First Come First Serve(FCFS) Scheduling**
- **Shortest-Job-First(SJF) Scheduling**
- **Priority Scheduling**
- **Round Robin(RR) Scheduling**

First-Come-First-Served algorithm

✓ Process that requests the CPU FIRST is allocated the CPU FIRST..

Other names of this algorithm are:

- **First-In-First-Out (FIFO) .**
- **Run-to-Completion .**
- **Run-Until-Done .**

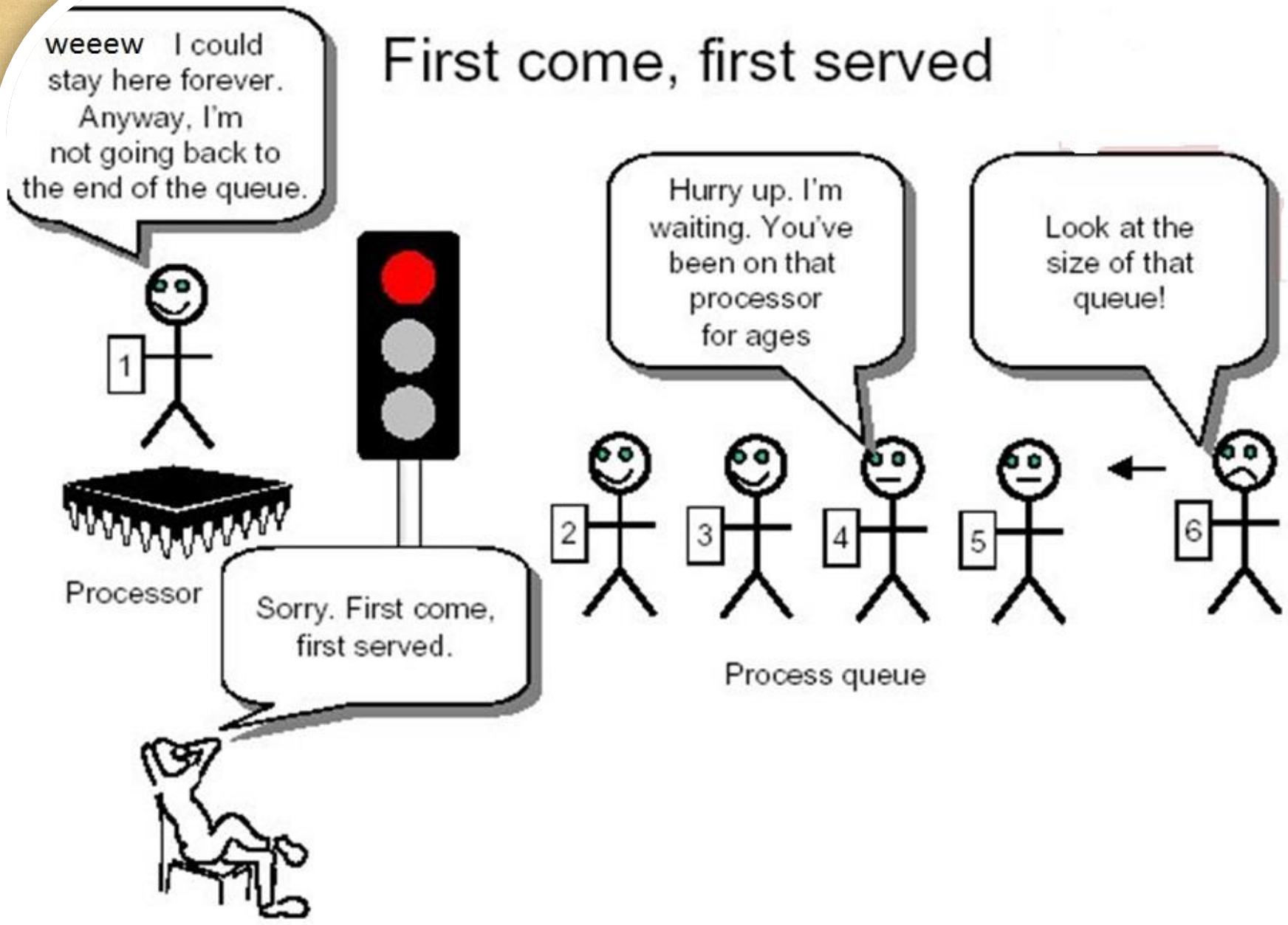
❑ **This is a non-preemptive algorithm.**

➢ Example on real life such as: Buying tickets !

➢ **Implementation**:

- **FIFO queues**.
- A new process enters the tail of the queue.
- The schedule selects from the head of the queue.

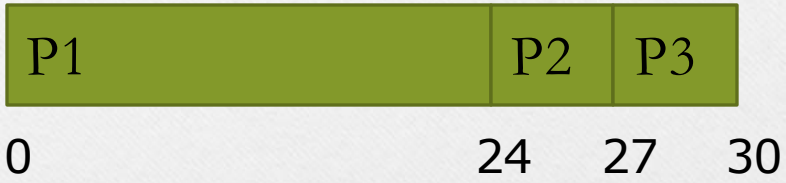
First come, first served



First-Come, First-Served (FCFS)

- Example: Three processes arrive in order P1, P2, P3.
  - P1 burst time: 24
  - P2 burst time: 3
  - P3 burst time: 3
- Waiting Time
  - P1: 0
  - P2: 24
  - P3: 27
- Completion Time:
  - P1: 24
  - P2: 27
  - P3: 30
- Average Waiting Time: (0+24+27)/3 = 17
- Average Completion Time: (24+27+30)/3 = 27

grantt chart



Advantages and Disadvantages:

Advantages:

- ✓Better for long processes
- ✓Simple method (i.e., minimum overhead on processor)
- ✓No starvation.

(Starvation occurs when a job cannot make progress because some other job has the resource it require)

Disadvantages:

- ○Non-preemptive.
- ○Not optimal AWT
- ✓**Convoy effect occurs**. Even very small process should wait for its turn to come to utilize the CPU. Short process behind long process results in lower CPU utilization.

Shortest job first (SJF)

✓Shortest job first (SJF) is a scheduling policy that selects the waiting process with the smallest execution time to execute next.

Other Names::

Shortest job next (SJN),

Shortest process next (SPN),

Shortest- Remaining-Time-First (**SRTF**).

Shortest job first (SJF)

Two schemes:

- ✓**Non-preemptive**-It is also known as Shortest Job First (**SJF**)
- ✓**Preemptive –** Also known as the Shortest-Remaining-Time-First (**SRTF**).

Non-preemptive SJF is *optimal if all the processes are* ready simultaneously– gives minimum average waiting time for a given set of processes.

SRTF is *optimal* *if the processes may arrive at different* times

Shortest job first (SJF)

Advantages:

✓Minimizes average waiting time.

✓Provably optimal w.r.t. average turnaround time

✓Throughput is high.

Disadvantages:

✓Requires future knowledge.

✓Elapsed time (i.e., execution-completed-time) must be recorded, it results an additional overhead on the processor.

✓Starvation may be possible for the longer processes.

Priority-Based 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).

•Priority scheduling can be either

- Preemptive
- Non-preemptive

• Equal-priority processes are scheduled in FCFS order.

• **Preemptive** - Preempt the CPU if the priority of the newly arrived process is higher than the priority of the currently running process.

Priority-Based Scheduling

•SJF is a priority scheduling where priority is the predicted next CPU burst time.

•Priority scheduling can suffer from a major problem known as **indefinite blocking, or starvation,**

Problem = **indefinite blocking or Starvation**

– low priority processes may never execute.

• **Solution** =**Aging**

– as time progresses increase the priority of the process

Priority-Based Scheduling

- **Advantage:**

- Good response for the highest priority processes.(**Non-preemptive**)

- Very good response for the highest priority process.(**preemptive**)

- **Disadvantage:**

- Starvation may be possible for the lowest priority processes.

Round-Robin Scheduling(RR)

This type of scheduling algorithm is basically designed for time sharing system.

✓It is **similar to FCFS** with preemption added.

✓Round-Robin Scheduling is also called as time-slicing scheduling and it is a preemptive version based on a clock.

✓Each process gets a small unit of CPU time (**time quantum**), usually 10-100milliseconds.

Performance

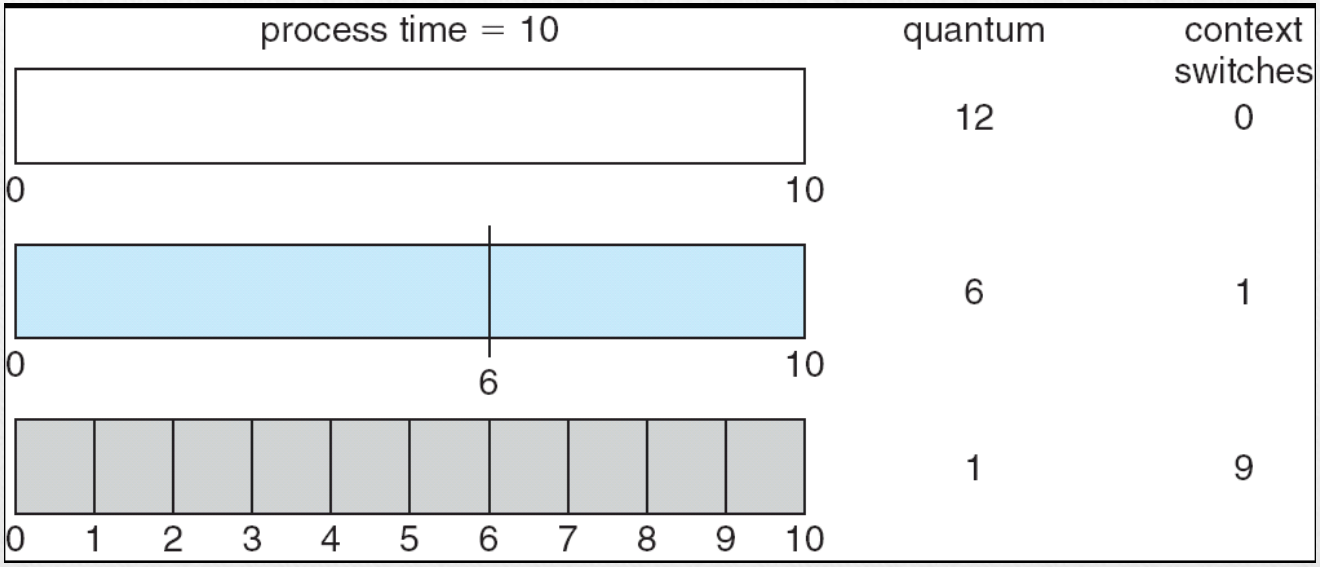
– **time quantum** *is too large =>FCFS*

-***time quantum*** *is too small => context switch overhead is too high*

Round-Robin Scheduling(RR)

Choosing a Time Quantum

- ✓The effect of quantum size on context-switching time must be carefully considered.
- ✓The time quantum must be large with respect to the context switch time
- ✓**Smaller time quantum increases context switches**



Round-Robin Scheduling(RR)

Advantages

- Round-robin is effective in a general-purpose, times-sharing system or transaction-processing system.
- Fair treatment for all the processes.
- Good response time for short processes.

Disadvantages

- Care must be taken in choosing quantum value.
- Processing overhead is there in handling clock interrupt.
- Throughput is low if time quantum is too small.

Inter-process Communication

- Process executing concurrently in the operating system may be either independent process or cooperating process.
- A process is independent if it cannot affect or be affected by the other process executing in the system. (does not share data)
- A process is cooperative if it can affect or be affected by the other processes executing in the system.(Shares data)

Inter-process Communication

- Reasons for providing an environment that allows process cooperation.
  - Information Sharing
  - Computing Speedup
  - Modularity
  - Convenience

Inter-process Communication

- Information Sharing
  - Several users may be interested in the same piece of information, we must provide an environment to allow concurrent access to such information.
- Computing Speedup
  - For faster execution the lager task is break into subtasks.

Inter-process Communication

- Modularity
  - Dividing the system functions into separate processes or threads
- Convenience
  - Even an individual user may work on many tasks at the same time. (Editing, Printing and Compiling in parallel)

Inter-process Communication

- Cooperating process requires an inter-process communication(IPC) mechanism that will allow them to exchange data and information.
- Two fundamental models of inter-process communication
  - Shared memory
  - Message passing

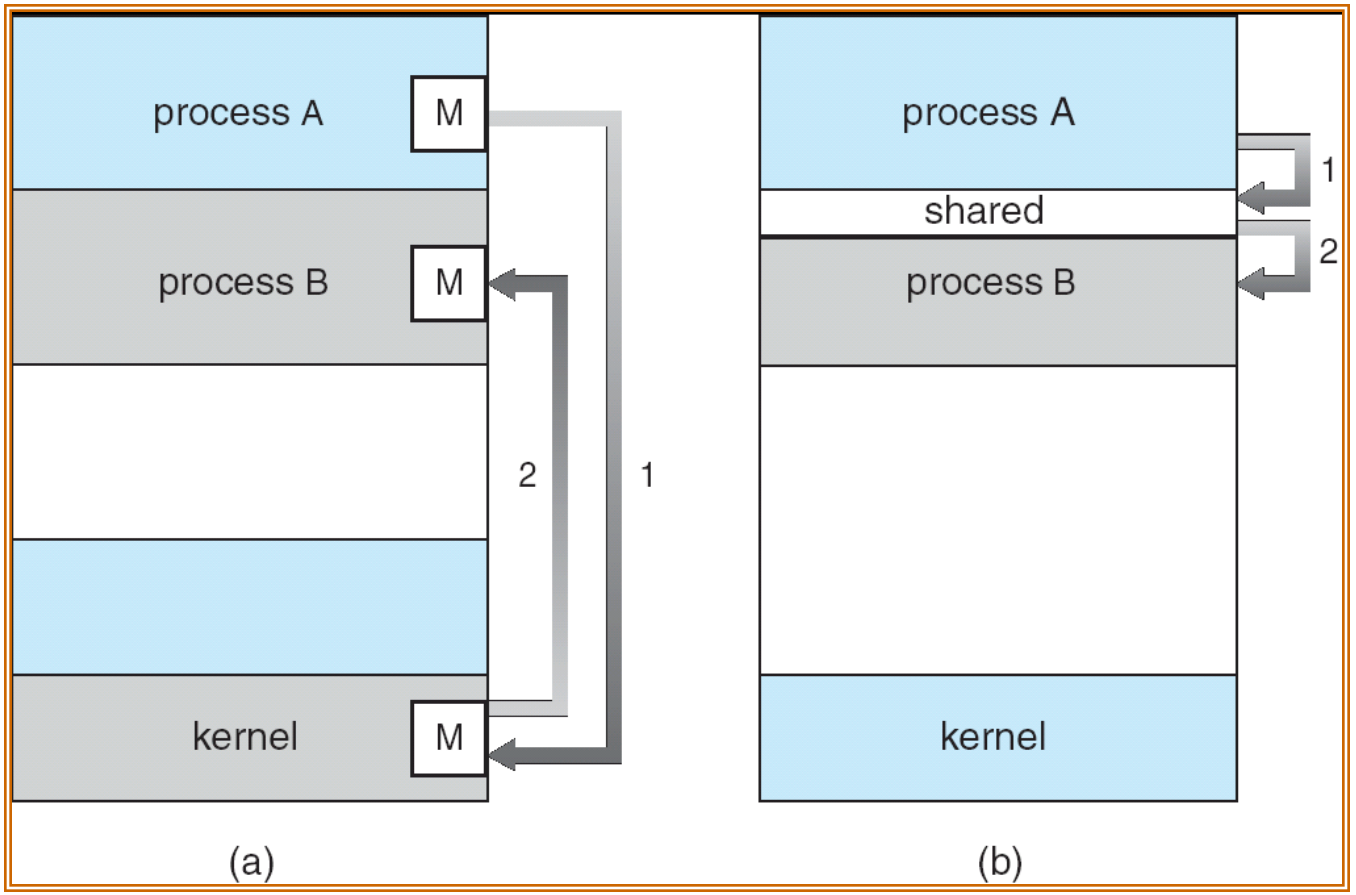
Interprocess Communication (IPC)

- 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*) – message size fixed or variable
  - **receive**(*message*)
- If *P* and *Q* wish to communicate, they need to:
  - establish a *communication link* between them
  - exchange messages via send/receive
- Implementation of communication link
  - physical (e.g., shared memory, hardware bus)
  - logical (e.g., logical properties)

Implementation Questions

- 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 of a link?
- Is the size of a message that the link can accommodate fixed or variable?
- Is a link unidirectional or bi-directional?

Communications Models



Interprocess Communication (IPC)

- Shared memory Systems
  - Establishes a region for shared memory
  - A shared memory region resides in the address space of the process creating the shared memory region.
  - Exchange data by reading and writing data
  - Ex: Producer Consumer problem

Interprocess Communication (IPC)

- Message-Passing Systems
  - Without sharing the same address space
  - Useful in distributed environment
  - Message passing facility provides send and receive messages.(fixed or variable size)
  - Communication link must be established

Interprocess Communication (IPC)

- Logical Implementation of Links
  - Direct or Indirect Communication
  - Synchronous or Asynchronous Communication
  - Automatic or Explicit Buffering

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 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 may be unidirectional, but is usually bi-directional

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 communication link
  - Link established only if processes share a common 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

Indirect Communication

- Operations
  - create a new mailbox
  - 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

Synchronization

- Blocking Send
- Nonblocking Send
- Blocking Receive
- NonBlocking Receive

Buffering

- Whether the communication is direct or indirect, messages exchanged by communicating processes resides in a temporary queue.
- Queues can be implemented in three ways
  - Zero capacity
  - Bounded capacity
  - Unbounded capacity