Process concepts

Process management

- A ***process*** -can be thought of as a program in execution.
- A process will need certain resources to accomplish its task Eg:- CPU time, memory, files, and I/O devices
- Resources are allocated to the process either when it is created or while it is executing
- A system mainly consists of 2 processes
    - Operating system processes
    - User processes

- Operating-system processes execute- system code
- User processes execute -user code
- These processes may execute concurrently

Process Concept

- Process may Include;
- Text section- more than of pgm code
- Data section – Global variables
- PCB-additional information
  - Program counter –contains address of the instruction to be executed, process registers
  - Process Stack- temp data eg: fn parameters, return address , local variables
  - Heap- memory which is dynamically allocated at run time

Program v/s process

- A program is a *passive* entity or static entity

  Eg: file containing a list of instructions stored on disk (often called an executable file)

- A process is an *Active* entity or Dynamic entity

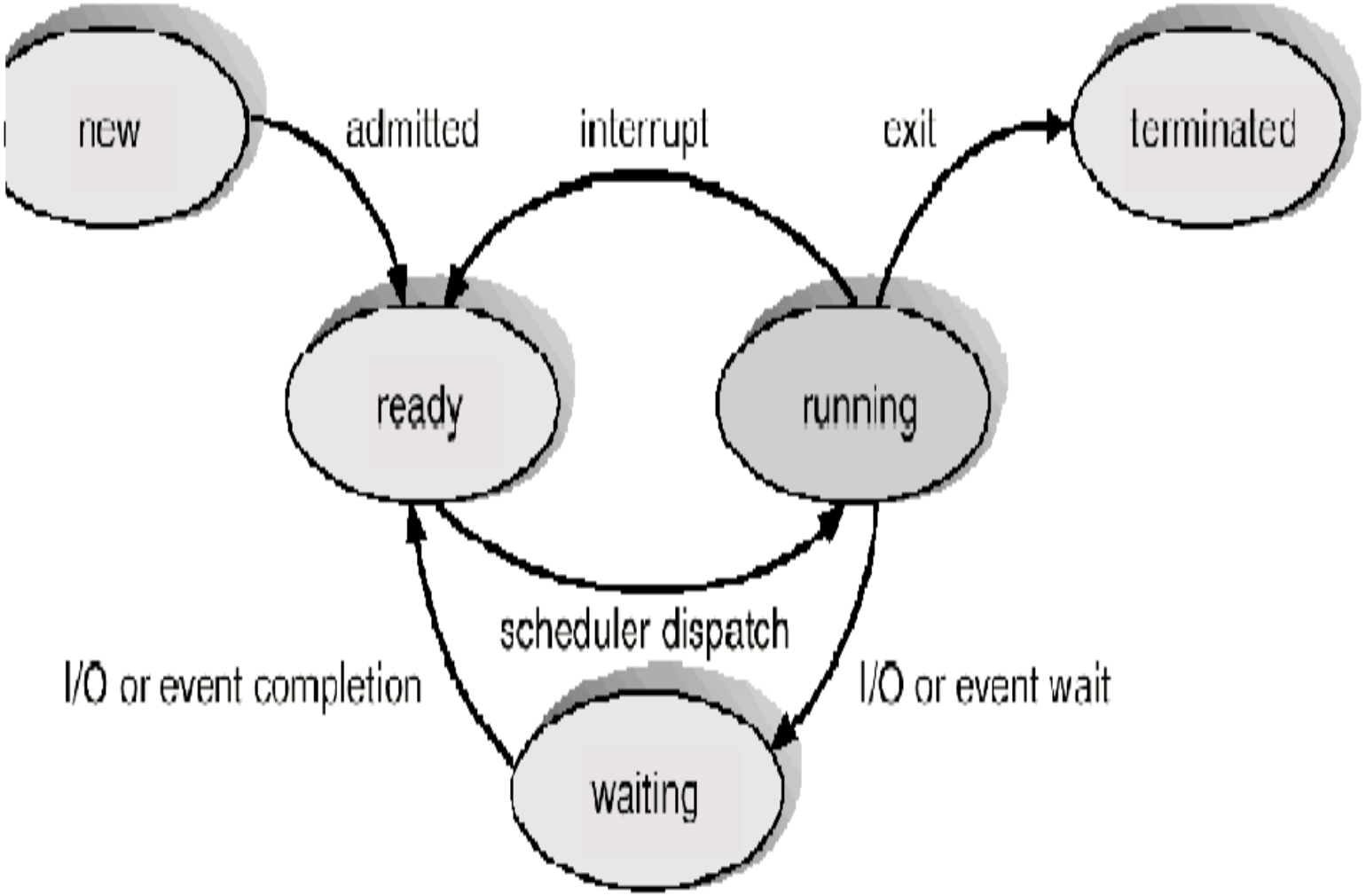
- Eg: program counter specifying the next instruction to execute and a set of associated resources .

- Note: A program becomes a process when an executable file is loaded into memory

Procces States

- As a process executes, it changes state.
- Process may be in one of the following states:
- New. A process is said to be in a ***New*** state if it is being created for the first time.
- Ready. The process is said to be in a ***ready*** if it is ready to execute when it gets the CPU.
- Running. The process is said to be in a ***running*** *state*, if CPU has been allocated to it and is running.
- Waiting. The process is waiting for some event to occur (such as an I/O or a signal).
- Terminated. The process has finished execution (either normally or abnormally)

Process state transition diagram



State transition

- New->ready
- Ready->running
- Running->ready
- Running->waiting
- Running->terminated
- Waiting->ready

- **Dispatcher**
  - Component in CPU scheduling
  - is a module that selects the process from the ready queue to allocate the cpu (ie.It is a module which gives control of the CPU to the process selected by short term scheduler)
  - It should be fast
  - The time taken to stop one process and start another is known as dispatch latency.
  - Its function involves
    - Switching context
    - Switching to user mode
    - Jumping to proper location in the user program to restart that program

Process Control Block (PCB)

- Process is represented in the operating system by a process control block (PCB)—also called a *task control block or* process descriptor.
- *PCB* is a data structure which contains descriptive information about a process
- Each process has an entry in the PCB
- PCB simply serves as the repository for any information that may vary from process to process.

PCB


process state
process number
program counter
registers
memory limits
list of open files
• • •

Information associated with PCB

1.Process state. – the state may be new, ready, running, waiting, halted, and so on.

2.Program counter. – it indicates the address of the next instruction to be executed for this process

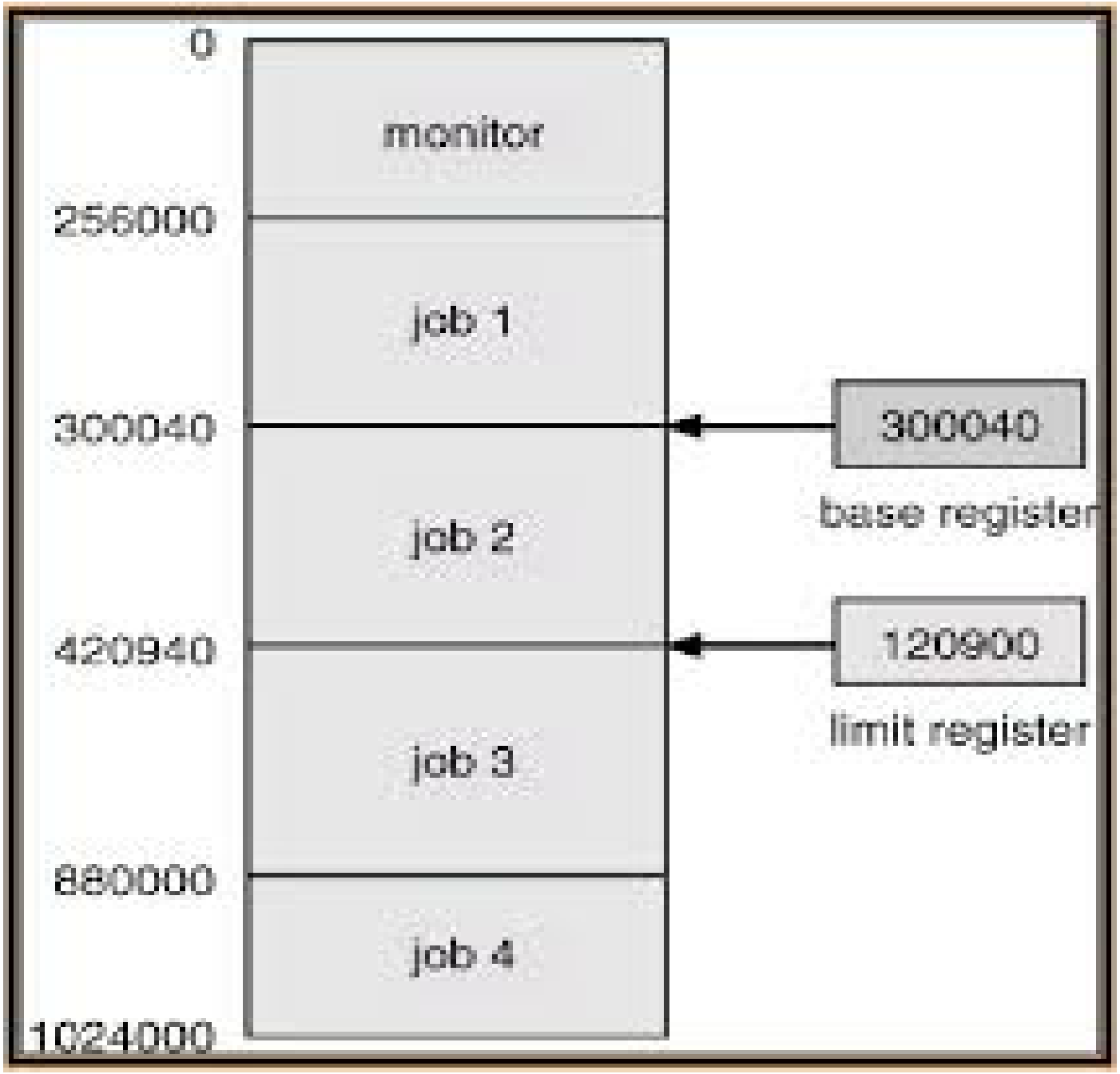
3.CPU registers.-when an interrupt occurs these details must be saved to continue its execution later. Eg: accumulators, index registers, stack pointers etc. The registers vary in number and type

4.CPU-scheduling information- information include process priority, pointers to scheduling queues, and any other scheduling parameters.

5.Memory-management information- this information include value of base and limit registers, the page tables, or the segment tables depending on the memory system used by the OS.

6.Accounting information- this information include the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on

7.I/O status information- this information include the list of I/O devices allocated to the process, a list of open files etc.


0
monitor
256000
job 1
300040
300040
base register
job 2
420940
120900
limit register
job 3
880000
job 4
1024000

Process Scheduling

- The objective of multiprogramming is to have some process running at all times to maximize CPU utilization.
- The objective of time-sharing is to switch the CPU among processes so frequently.
- The process scheduler selects an available process (possibly from a set of several available processes) for program execution on the CPU.

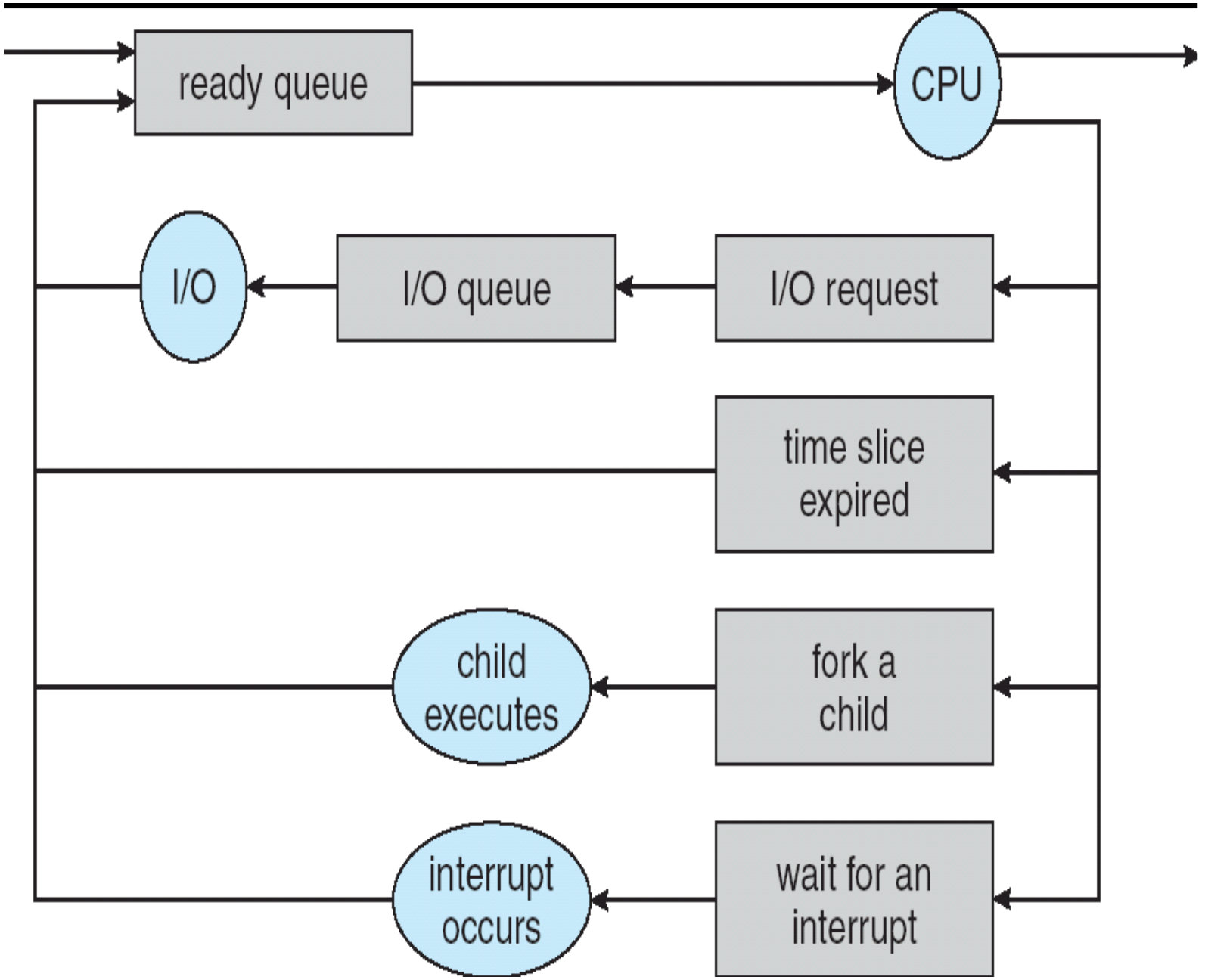
Scheduling Queues

- **Job Queue**
  - it consists of all processes in the system
  - When a new process enters the system it is put in the job queue
- **Device Queue**
  - The list of processes waiting for a particular I/O device . Each device has its

- ## Ready queue

  - It contains a list of processes waiting in main memory which are ready to execute when it gets the CPU.
  - It is stored as linked list.
  - It has a ready queue header which contains links to first and final PCB's in the list
  - Each PCB has a pointer field which points to the next PCB in the ready queue.

- ***Queuing diagram is shown below : -***


ready queue
CPU
I/O
I/O queue
I/O request
time slice expired
child executes
fork a child
interrupt occurs
wait for an interrupt

Once the process is allocated the CPU and is executing, one of several events could occur;

a) The process could issue an I/O request and then be placed in an I/O queue(device queue).

b) If the time slice(time sharing systems) of the process has expired, the process is put back to ready queue

c) The process could create a new sub process and wait for the sub process's termination (stack).

d) The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.

A process continues this cycle until it terminates, at which time it is removed from all queues and has its PCB and resources de-allocated

Schedulers

- A process migrates among the various scheduling queues .
- Operating system must select processes from these queues in some fashion.
- Selection process is carried out by the appropriate scheduler.
- Processes are spooled to a mass-storage where they are kept for later execution.

Types of schedulers

- Long-term scheduler
- Medium-term scheduler
- Short-term scheduler

Long-term scheduler

- The long-term, or admission scheduler, selects processes from job pool and loads to the Main Memory for execution(**ready queue**).
- Executes less frequently & can take more time to select a process
- Selection should be careful with a mix of both CPU-bound and I/O-bound processes.
- Controls the degree of multiprogramming(no of processes in main memory)
- It is invoked only when a process exits from the system

Medium term scheduler (also known as swapper)

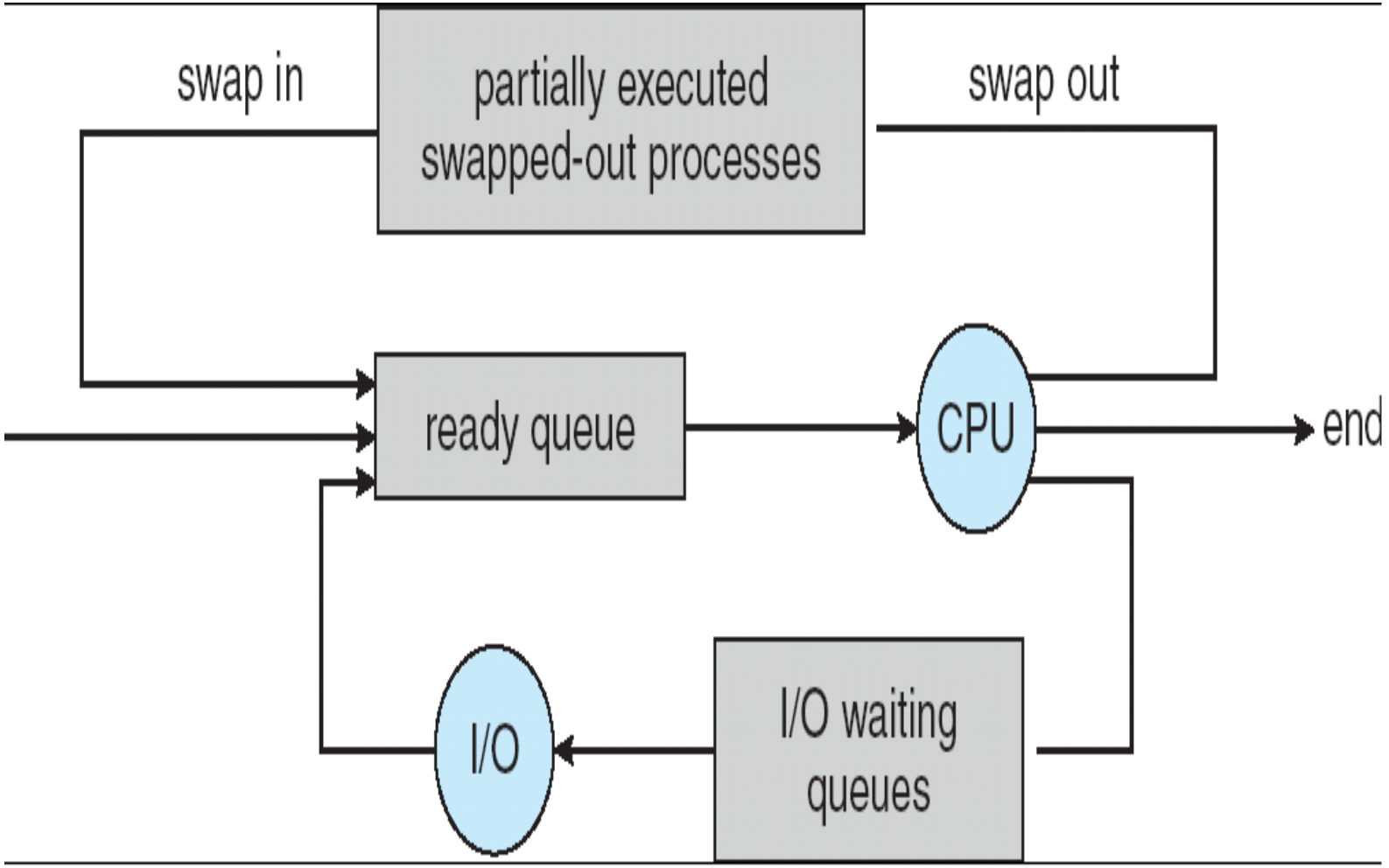
- The medium-term scheduler **temporarily removes** processes from main memory and places them on hard disk drive and later the process can be brought back to the memory and can continue its execution from where it left off. This scheme is called swapping

- This is commonly referred as "swapping out" or " swapping in"

- which determines when processes are to be suspended and resumed.

-sometimes it can be advantageous to remove processes from memory to avoid deadlock or to improve the process mix

Medium term scheduler- swap out & swap in



- # Short-term scheduler

- The short-term scheduler (also known as the CPU scheduler) selects a process from the memory which is ready for execution(from ready queue) and allocates CPU to it.

- It selects a new process more frequently than long-term as the process executes for a very short time before waiting for an I/O request

- So short term scheduler must be fast

- CPU scheduling takes place under the following conditions
  - When a process switches from running to waiting state (I/O event or ..)
  - When a process switches from running to ready state (interrupt occurs)
  - When a process switches from waiting to ready state (I/O completion)
  - When a process terminates

Context Switch

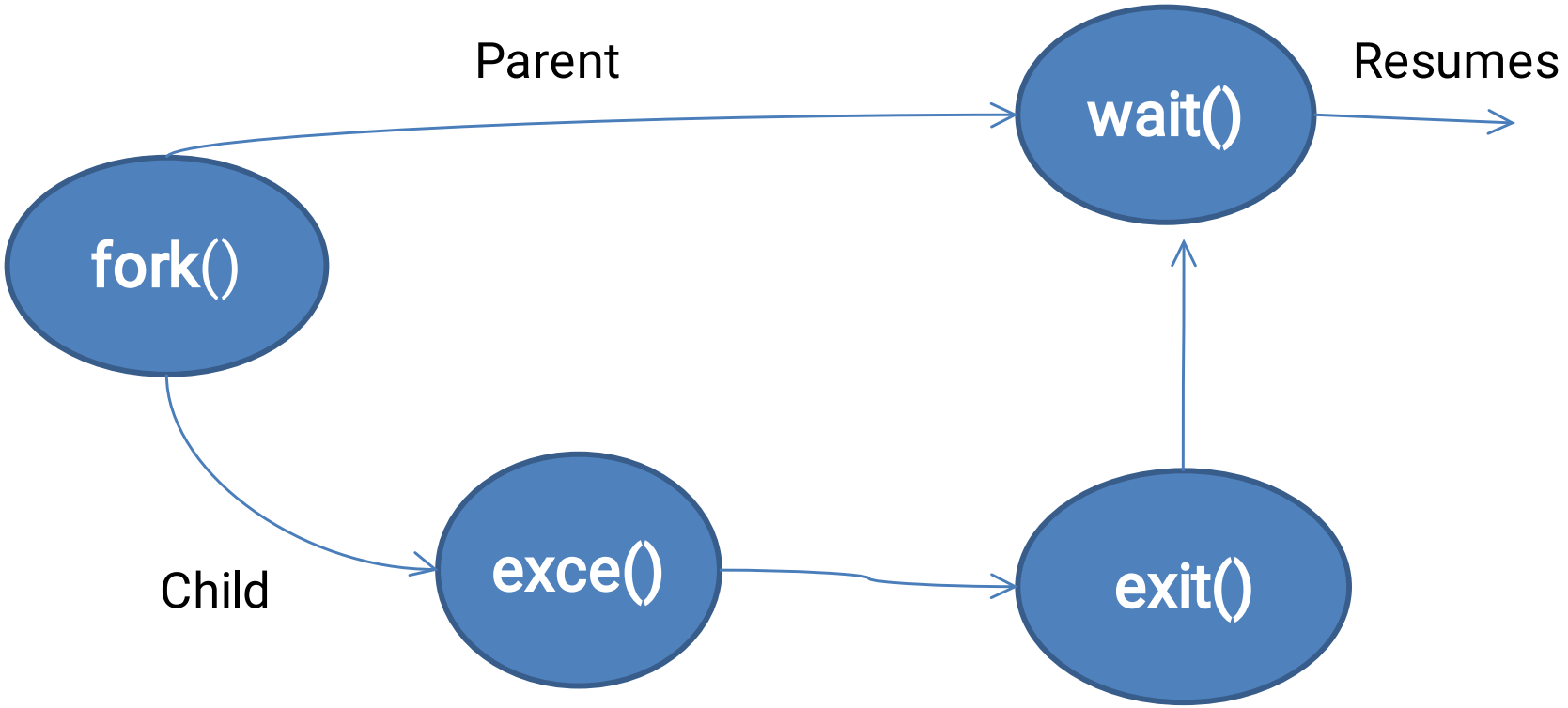
- Switching the CPU from one process to another requires saving the state of the old process and loading the saved state for the new process. This task is known as Context Switch.
- Context of a process is represented in its PCB
- When a context switch occurs , the kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.
- Context switch times are hardware dependent
- It is a 2 step process
  - Save the context – to save PCB of current process
  - Restore the context – restore PCB of old process to resume operation

Operations on processes

- **Process Creation**
  - A process (parent) can create many new processes(child) via *create-process*s System call (**fork()**) and a child process in-turn can create other processes forming a tree.
  - Each process is identified using a unique processID which is an integer value.
  - A child process can obtain its resources directly or from its parent
  - A parent has to share its resources among its children
  - A child can have same or separate program and data of its parent
  - When a child process is created, there are 2 possibilities

- New process is created by fork() and the fork() returns **'0'** to the child and ***child ID*** to the parent.
- ***Exec()*** replaces the entire current process with a new program.

Process creation



- **Process Termination**
  - A process terminates when it finishes executing its final statement and asks the OS to delete it by using the **exit()** system call
  - At this time the child process returns the status(PID of a terminated process) value to its parent process.
  - Now all the resources are de-allocated by the OS
  - A process can terminate another process via **TerminateProcess()** system call.

- **A parent can terminate its children when**
  - A child exceeded its resource usage
  - The task assigned to the child is no longer required
  - The parent is exiting (cascading termination)

Inter process Communication

2 types of processes

1. Independent processes - cannot affect or be affected by the other processes executing in the system, they do not share data with other processes

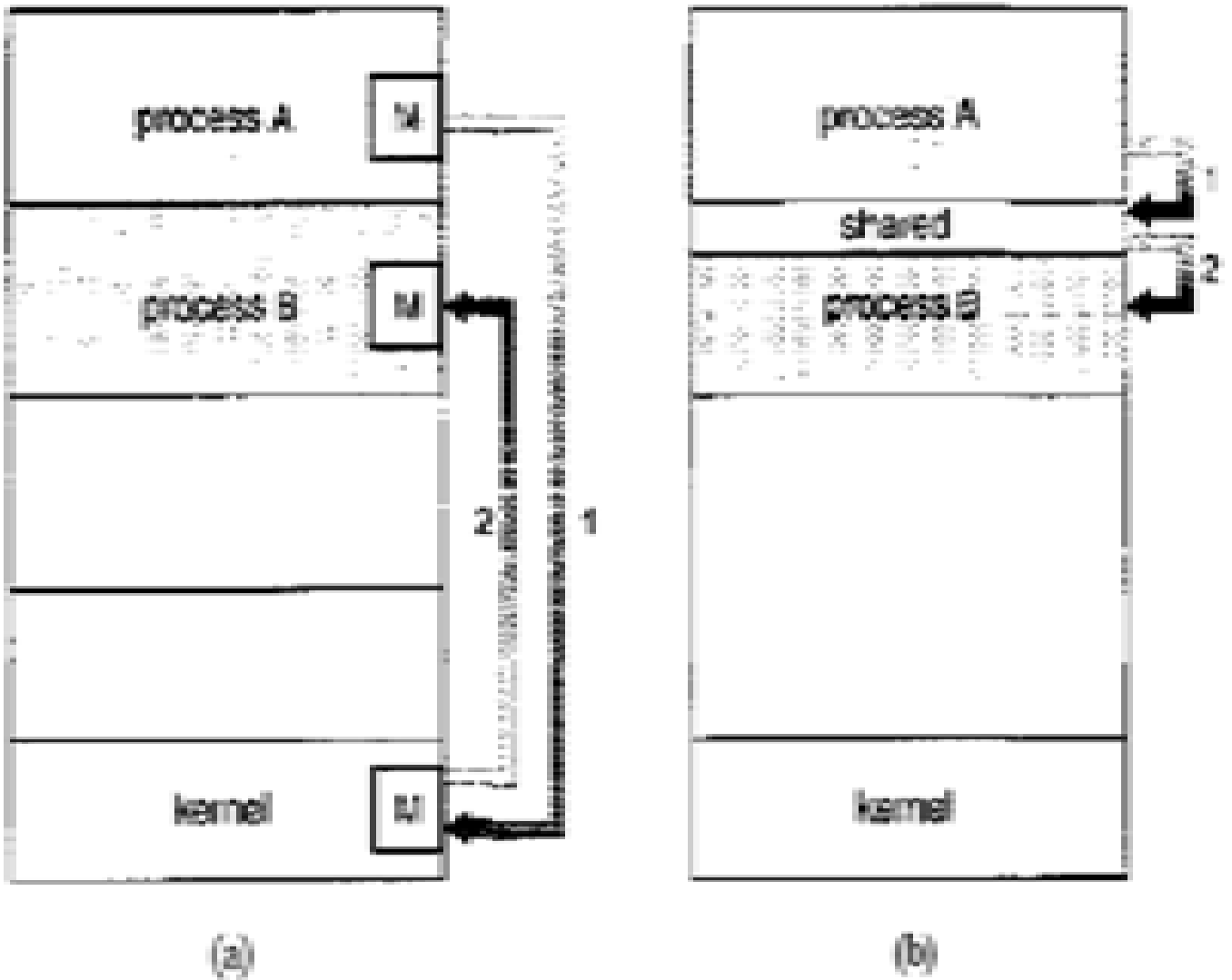
2. Cooperating processes - it can affect or be affected by the other processes executing in the system, they share data with other processes

- **Reasons for process cooperation**
  - **Information Sharing** – when many users are interested same piece of information, simultaneous access to that information is needed
  - **Computation Speedup** – tasks can run faster by breaking one task into subtasks and these sub tasks are executed parallel provided that there is multiple processing elements(CPU or I/O channels)
  - **Modularity** – system should be in modular fashion, dividing the functions into separate processes or threads
  - **Convenience** – individual user can work on many tasks at the same time like editing, printing, compiling etc. in parallel

- There are two fundamental models of inter process communication:

(1) Shared memory - a region of memory that is shared by cooperating processes .

(2) Message passing - communication takes place by means of messages exchanged between the cooperating processes with OS intervention.



ure 3.13 Communications models. (a) Message passing. (b) Shared memory.

CPU scheduler(short-term scheduler)

- It allocates process from main memory(ready queue) to the CPU using various scheduling algorithms.

- 2 types of scheduling
  - Preemptive – it is prioritized. A running process can be stopped forcibly by sending interrupts when a high priority process comes or due to some other reasons. In this, Good mechanisms should be used to coordinate access to shared data.
  - Non-preemptive - once CPU has been allotted for a process, the process keeps the CPU until it releases either by terminating or by switching to a waiting state.

Scheduling Criteria (in selecting CPU scheduling algorithms)

i) CPU utilization– keep the CPU as busy as possible. CPU utilization should be in the range of 0 to 100 percentage.

ii) Throughput– no: of processes that complete their execution per time unit. For long processes it may be 1 process/hr and for short processes it may be 10 process/second.

iii) Turnaround time – amount of time b/n the submission and completion of a process. It is the sum of time spent waiting to get memory, waiting in the ready queue, executing on CPU and doing I/O.

iv) Waiting time - amount of time a process has been waiting in the ready queue. CPU scheduling algorithm is affected by this waiting time.

v. Response time– amount of time it takes from when a request was submitted until the first response is produced, not output (time it starts responding). It is limited by the speed of the output device

- **Scheduling algorithms are selected in way such that**
  - To maximize CPU utilization
  - To maximize Throughput
  - To minimize Turnaround time
  - To minimize Waiting time
  - To minimize Response time
- Scheduling deals with the problem deciding which of the processes in the ready queue can be allocated to the CPU

Scheduling Algorithms

1) First-Come, First-Served (FCFS) Scheduling

2) Shortest-Job-First Scheduling (SJF Scheduling)

3) Priority Scheduling

4) Round-Robin Scheduling

5) Multilevel Queue Scheduling

6) Multilevel Feedback-Queue Scheduling

First-Come, First-Served (FCFS) Scheduling

- Process are executed in the order of their arrival in the ready queue
- It is a non-preemptive scheduling algorithm
- So once the CPU is allocated to the process, it retains control of CPU until it blocks(waiting state) or terminates
- Here ready queue is managed in FIFO manner
- Average waiting time and turnaround time can be reduced if the processes having shorter CPU burst execute before those having longer CPU burst.

Example problems

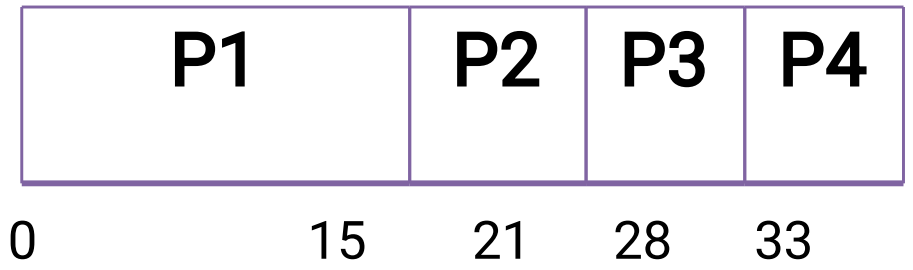
- **Problem 1: (Ans : AWT=13.5ms ATAT=21.75ms)**

Process	P1	P2	P3	P4
Arrival time	0	2	3	5
CPU burst(ms)	15	6	7	5

- **Problem 2: (above problem with change in arrival time of processes) : (Ans : AWT=6ms ATAT=14.25ms)**

Process	P4	P2	P3	P1
Arrival time	0	2	3	5
CPU burst(ms)	5	6	7	15

Solutions for problem1



Waiting Time for P1 = 0

15-0=15

Waiting Time for P2 = 15-2 =13

Waiting Time for P3 = 21-3=18

Waiting Time for P4 = 28-5 = 23

Average Waiting time =

(0+13+18+23)/4=**13.5**

Turn around time for P1 =

Turn around time for P2 =

21-2=19

Turn around time for P3 = 28-3 =

25

Turn around time for P4 = 33-5

= 28

Average Turn Around time =

(15+19+25+28)/4 **= 21.75**

- **Advantages**
  - It is easy to understand, implement and manipulate as processes are added only at the end and removed only from the front of queue.
  - Well suited for batch systems where longer time periods for each process are often acceptable.
- **Disadvantages**
  - Average waiting time varies from process to process. This is not recommended where performance is a major issue
  - It reduces CPU and I/O utilization in some instances. There is a convoy effect as all the other processes wait for the one big process to get off the CPU.
  - Not suitable for time sharing systems where each process should get equal amount of CPU time

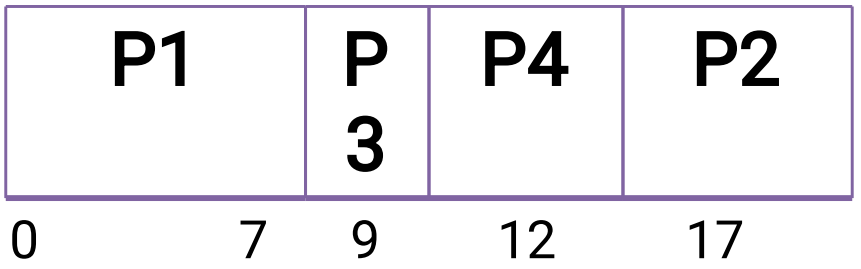
SJF Scheduling

- Non-preemptive
- According to the length of CPU burst time
- Among the ready processes, process with shortest execution time(CPU burst time) is selected first
- Here longer process has to wait until all processes shorter than it has been executed
- If two processes has the same CPU burst then they are scheduled in FCFS order

Process	P1	P2	P3	P4
Arrival Time	0	1	3	4
CPU burst time(ms)	7	5	2	3

AWT = 5
Att=9.25

solution



WT for P1 = 0-0=0 TAT for p1 = 7-0=7

WT for P2 = 12-1=11 TAT for p2=17-1=16

WT for P3=7-3=4 TAT for p3=9-3=6

WT for P4=9-4=5 TAT for p4=12-4=8

Average WT = Average TAT =

(0+11+4+5)/4 = **5ms** (7+16+6+8)/4=**9.25ms**

Advantages

- Eliminates the variance in waiting time and turn around time

Disadvantages

- It is little complicated to implement as the exact length of CPU burst should be known in advance
- it results in starvation of long processes

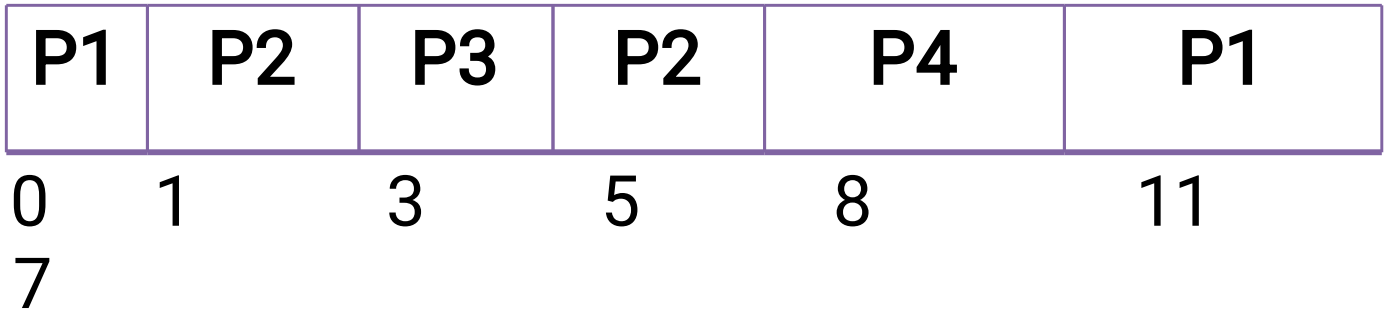
Shortest-Remaining-Time-First Scheduling

- SJF can be either preemptive or non-preemptive
- This is a type of preemptive SJF scheduling
- If a new process have a short CPU burst than the left CPU burst time of currently executing process, preemptive SJF scheduling will preempt the currently running process.

Process	P1	P2	P3	P4
Arrival Time	0	1	3	4
CPU burst time	7	5	2	3

AWT = 4
Att=8.25

Solution for the above problem : Gantt Chart



WT for p1 = (0-0)+(11-1)=10

WT for p2 = (1-1)+(5-3)=2

WT for p3=3-3=0

WT for p4=8-4=4

Average WT =

(10+2+0+4)/4 = 4

TAT for p1 = 17-0=17

TAT for p2 = 8-1=7

TAT for p3=5-3=2

TAT for p4=11-4=7

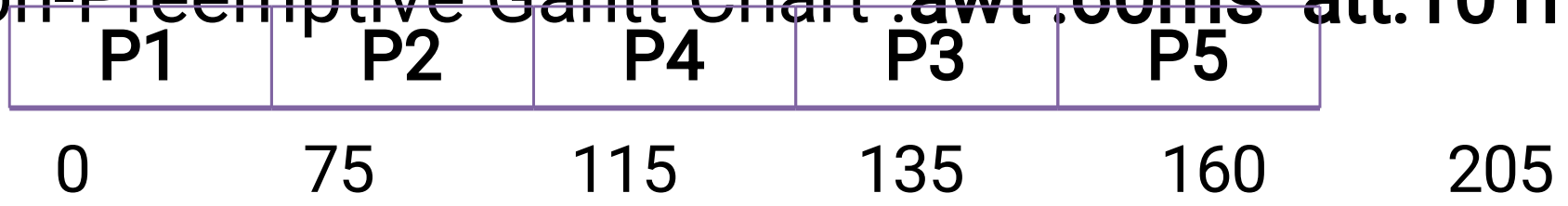
Average TAT =

(17+7+2+7)/4 = 8.25

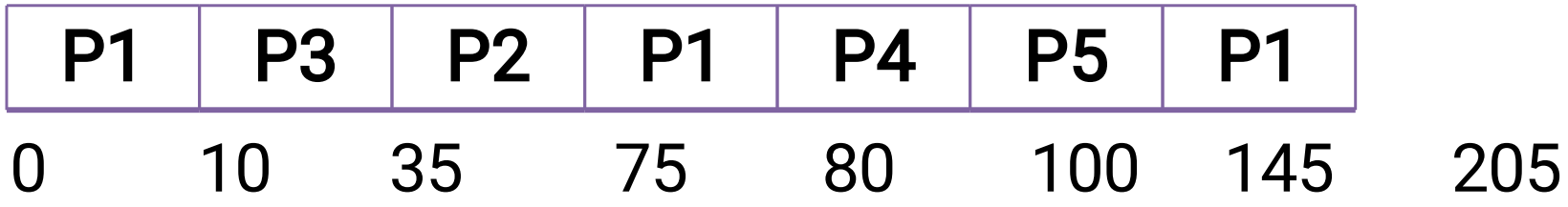
- **Important Problem (both preemptive and non-preemptive SJF)**

Process	P1	P2	P3	P4	P5
Arrival Time	0	10	10	80	85
CPU burst time	75	40	25	20	45

Non-Preemptive Gantt Chart :**awt :60ms att:101ms**



Preemptive Gantt Chart :**awt :32ms att:75ms**



Advantages

- A long process near to its completion may be favoured over short processes entering system resulting in an improvement in turn around time

Disadvantages

- Like SJF, it Requires to know the next CPU burst in advance
- Here starvation of long process may occur.

Priority Scheduling

- SJF is a special case of general priority-scheduling algorithm (i.e. larger the CPU burst lower the priority)
- Process with highest priority will be scheduled first
- If two process with same priority comes, they are executed in FCFS order
- It can be either preemptive or on-preemptive
- Priorities can be defined either internally(based on measurable qty) or externally (done by OS based on importance, type etc.)
- Here assume low numbers represent high priority

Advantages

- Important processes are executed first

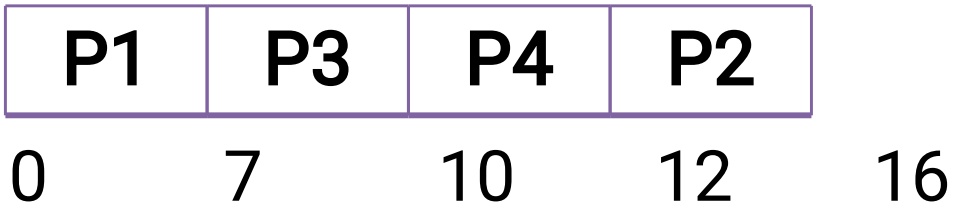
Disadvantages

- Lower priority process suffers from starvation (sometimes low priority processes waits indefinitely for the CPU).
- One solution for the above problem is **aging -** as time progresses increase the priority of the process.

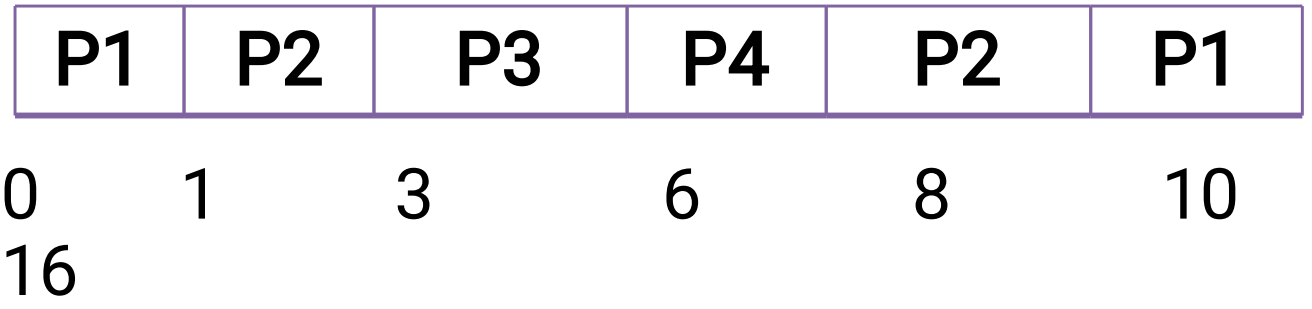
- Problem 1(both preemptive and non-preemptive)

Process	P1	P2	P3	P4
Arrival Time	0	1	3	4
CPU burst time	7	4	3	2
Priority	4	3	1	2

Solution - Non-Premptive **AWT** :5.25 ATA:9.25



Solution - Premptive **AWT** :4 ATA:8

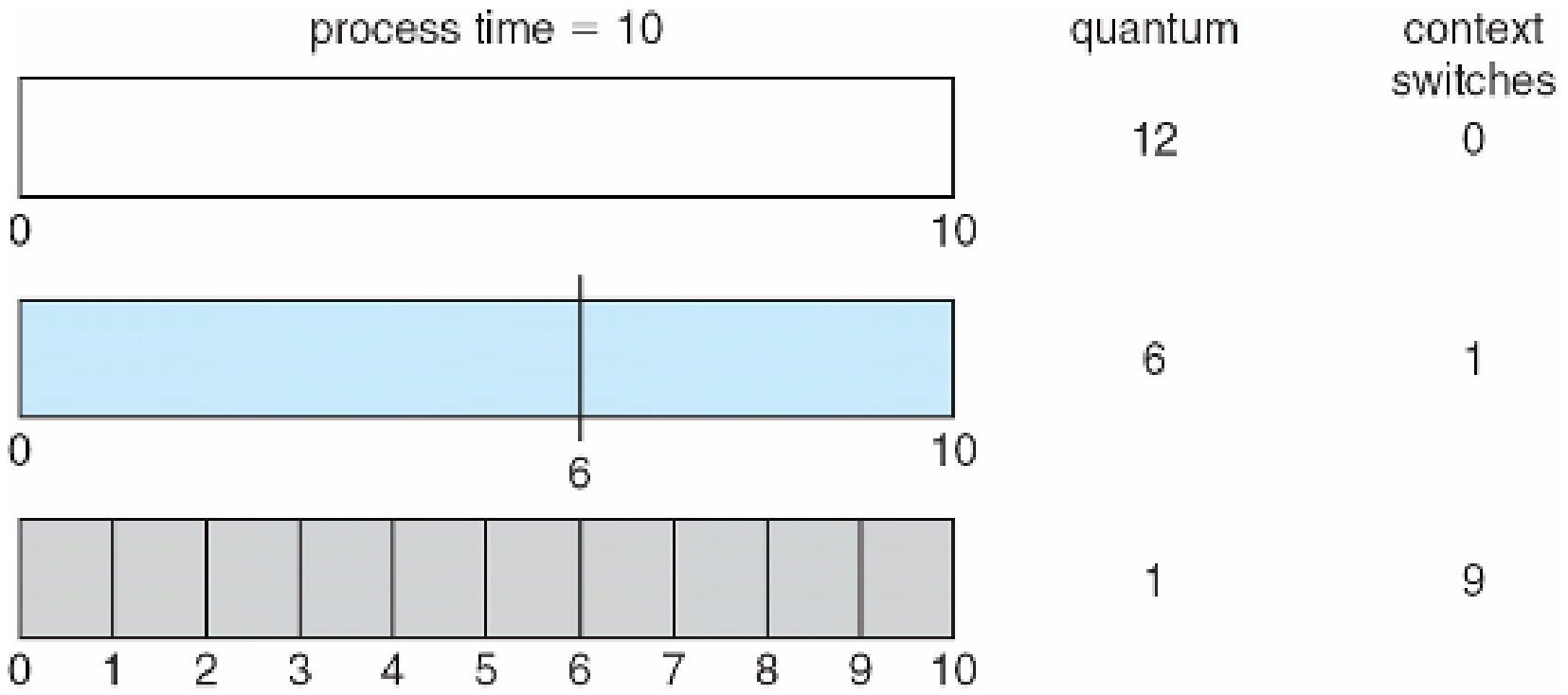


Round-Robin Scheduling

- Each process in the ready queue gets a fixed amount of CPU time (*time quantum or time slice*), usually 10-100 milliseconds.
- After this time has elapsed, the process is preempted and added to the end of the ready queue.
- Designed mainly for time sharing systems.
- Ready queue is treated as circular queue
- RR scheduling algorithm is preemptive

- **Advantages**
  - Efficient for time sharing systems
  - Increases fairness among processes
- **Disadvantages**
  - Even short processes may take long time to execute
  - Requires extra hardware support such as timer to interrupt after each time slice

Time Quantum and Context Switch Time

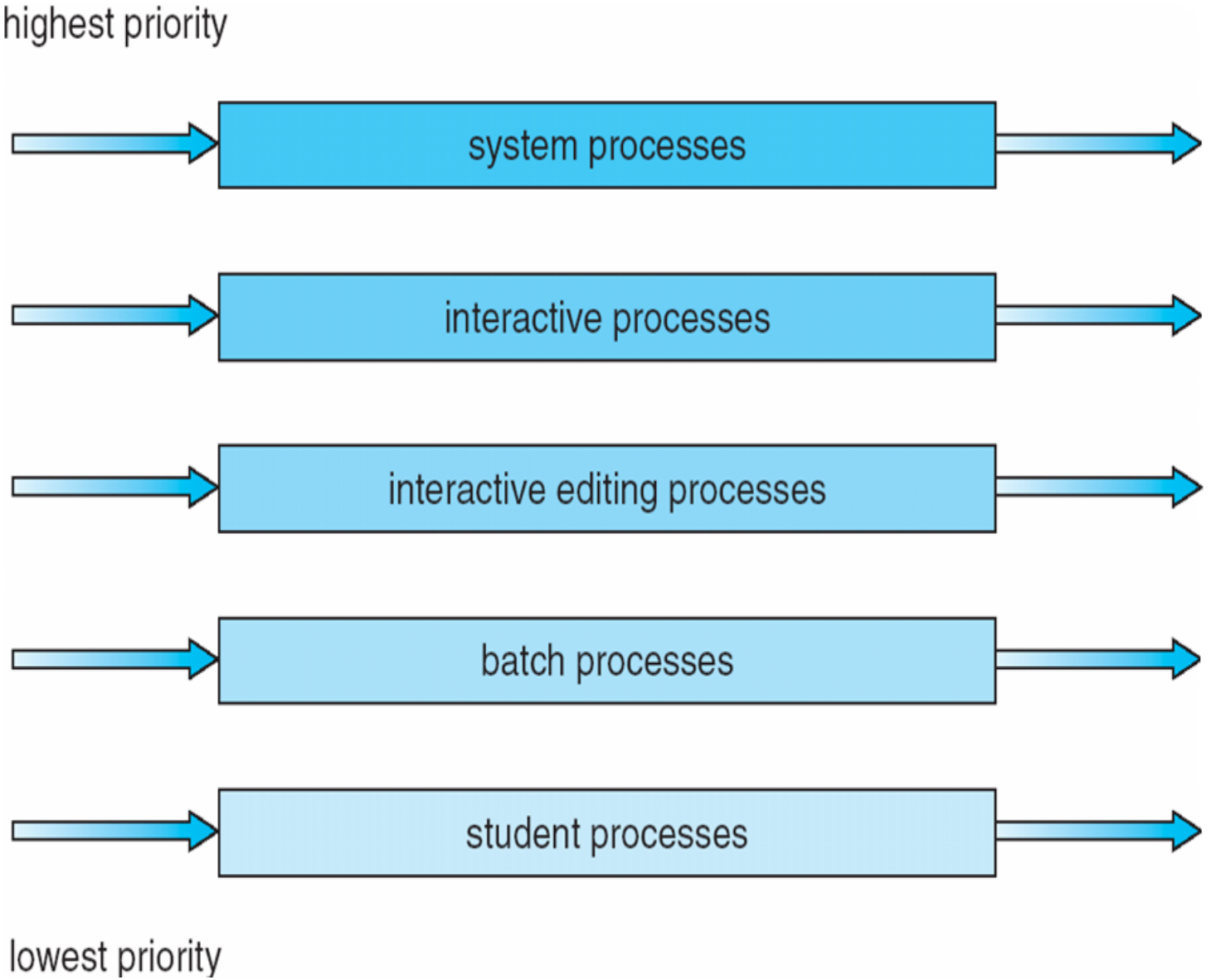


Showing how smaller time quantum increase context switches

Multilevel Queue

- It partition the Ready queue into several separate queue. Eg : foreground (interactive) & background (batch) as they have different response time requirements .
- Processes are **permanently** assigned to one queue depending on memory size, priority, process type etc.
- Each queue has its own scheduling algorithm
  - foreground – RR
  - background – FCFS

Multilevel Queue Scheduling



Scheduling must be done between the queues as

- Fixed priority preemptive scheduling
  - Each queue has absolute priority over lower priority queues. (i.e., serve all from foreground then from background).
  - If a process **p1** enters in a higher priority queue while a process **p2** in a low priority queue is executing, then **p2** is preempted and **p1 st**arts executing.
  - There is a possibility of starvation of low priority processes.
- Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes;

Multilevel Feedback Queue scheduling

- This Allows a process to move between queues
- Useful to separate process with different CPU burst features
- If a process use too much CPU time it is moved to lower priority queue
- A process that waits for long time is the lower priority queue can be moved to a higher priority queue using **aging** & can prevent the starvation.
- It is the most general and most complex CPU scheduling algorithm.

- **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 in which queue a process will enter when that process needs service

Multiple-Processor Scheduling

- CPU scheduling more complex when multiple processors(CPUs) are available, but load sharing can be done between these multiple processors.
- 2 types of processors

  - **Homogeneous processors-** identical functionality

  - **Heterogeneous processors** - different functions and properties

- Approaches to Multiprocessor Scheduling are :
  - **Symmetric multiprocessing (SMP)** –
    - **Each processor is self-scheduling**
    - Each has its own private queue
    - Ensure 2 processors do not select same process
  - **Asymmetric multiprocessing(AMP)**
    - **master-slave approach**
    - only one processor(master) accesses the system data structures, alleviating the need for data sharing

- **Load balancing**
  - It tries to keep workload evenly distributed across all processors in an SMP system
  - It is important in systems with s**eparate queue as when** one processor is busy, others may be idle
  - In **common ready queue systems** every one gets equal load
  - Load balancing has **push migration (**moving process from a busy processo**r)** and **pull migration**(idle processor pulls a waiting process from a busy processor**)**

- **Processor Affinity**
  - It means an effort to make a process to run on the same processor it was executed last time so that the cache memory of a processor can be used and the execution can be made faster.
  - Soft affinity – no guarantee, OS tries to allocate same processor
  - Hard affinity – forces a process to run on same processor