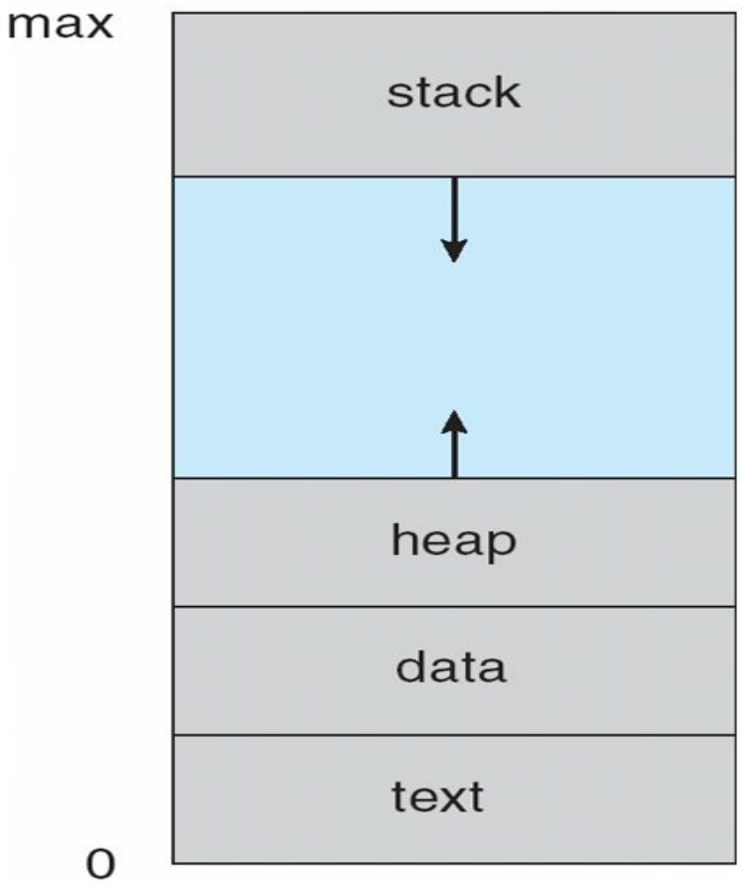
Process concept

- Process : is a program in execution. It is a active entity where as program is passive entity.
- 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 also 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. The structure of a process in memory is shown in Figure.

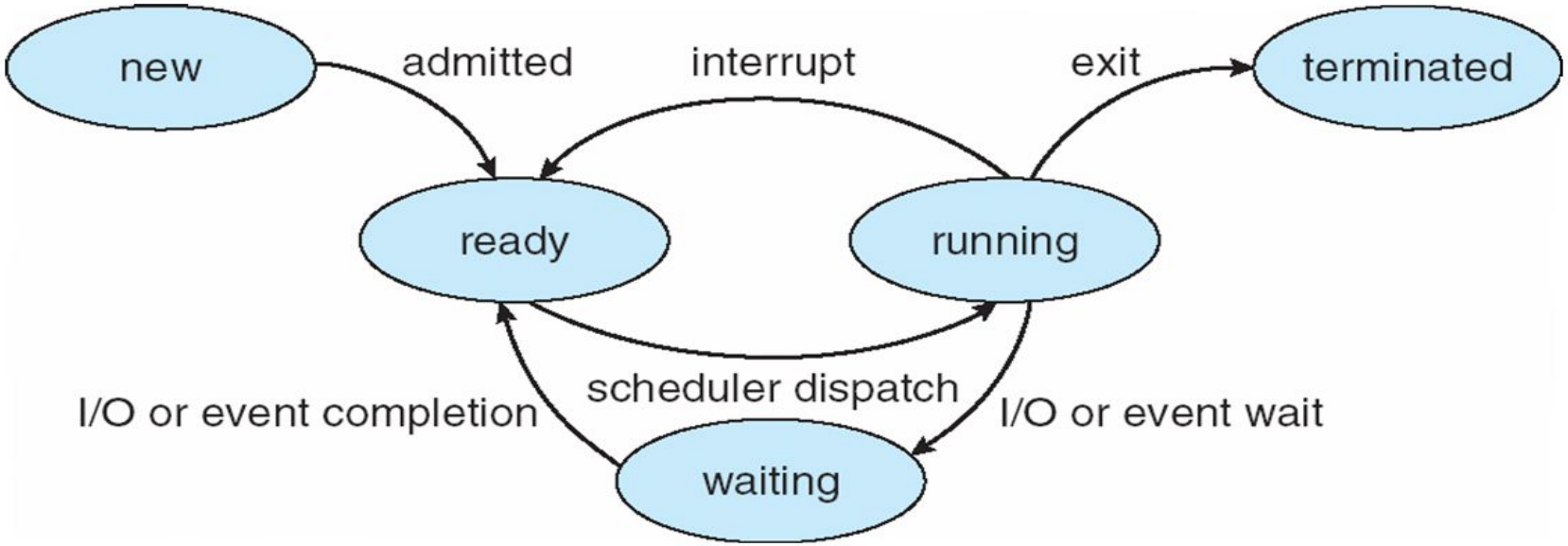
Continued…

- Process in memory



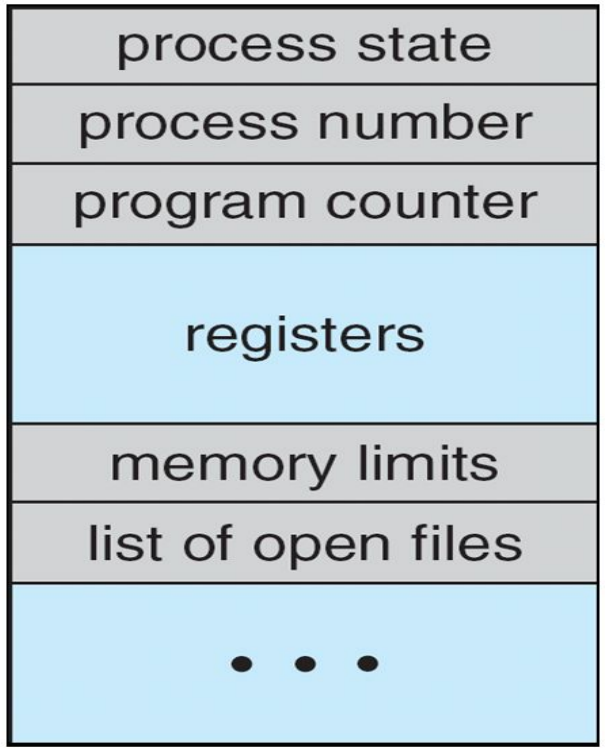
Process state

- 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
- It is important to realize that only one process can be *running* on any processor at any instant. Many processes may be *ready* and *waiting.*



Process Control Block

- Each process is represented in operating system by a process control block also called as task control block.



Continued….

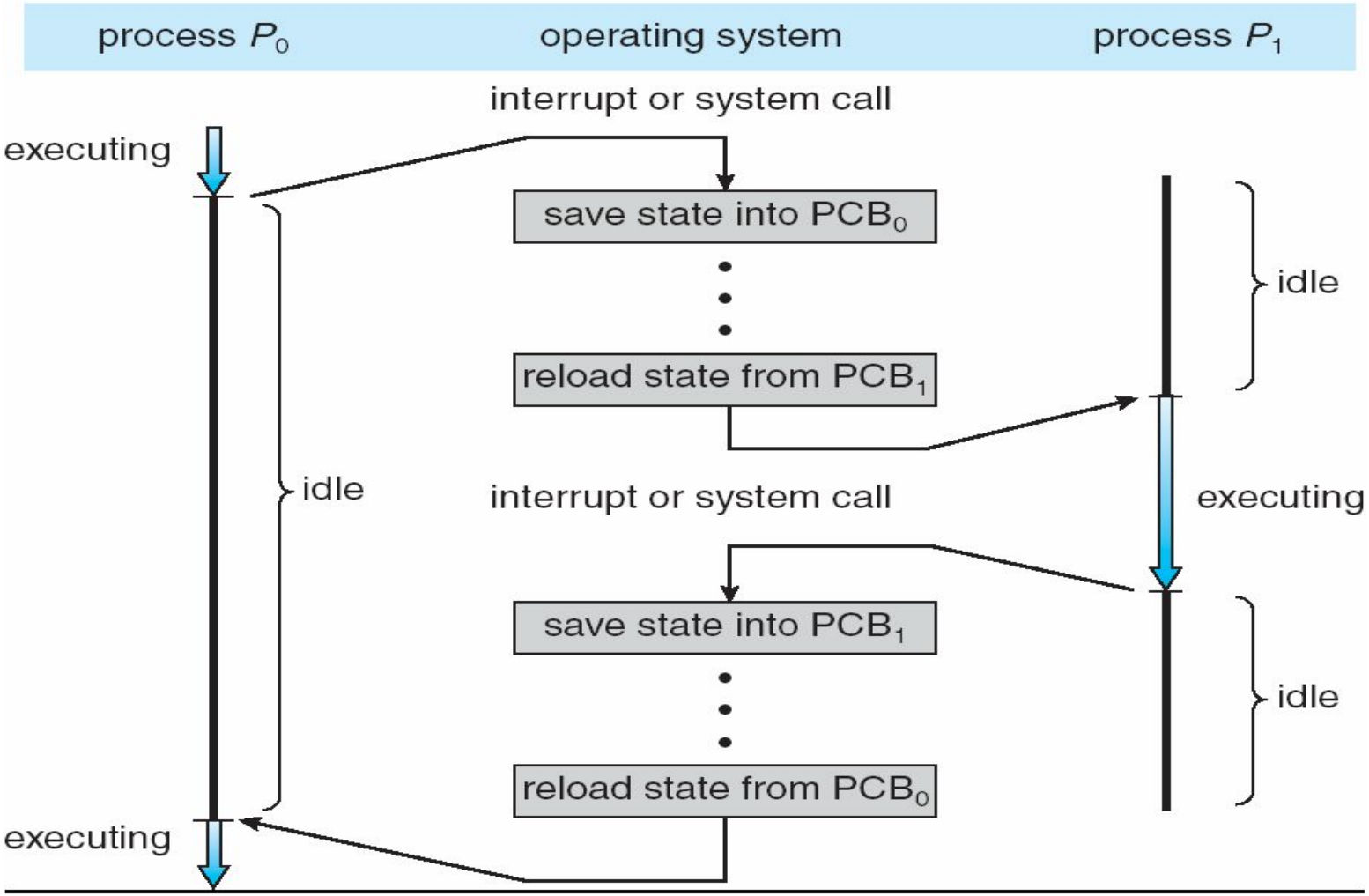
- Process state: The state may be new, ready, running, waiting, halted, and so on.
- Program counter: The counter indicates the address of the next instruction to be executed for this process.
- CPU registers: The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code information. Along with the program counter, this state information must be saved when an interrupt occurs.
- 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.

Threads

- The process model discussed so far has implied that a process is a program that performs a single thread of execution. For example, when a process is running a word-processor program, a single thread of instructions is being executed. This single thread of control allows the process to perform only one task at one time. The user cannot simultaneously type in characters and run the spell checker within the same process, for example. Many modern operating systems have extended the process concept to allow a process to have multiple threads of execution and thus to perform more than one task at a time.

Continued..

CPU Switch From Process to Process



3.2 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 that users can interact with each program while it is running. To meet these objectives, the process scheduler selects an available process (possibly from a set of several available processes) for program execution on the CPU.

3.2.1 Scheduling Queues

- **Process scheduler** selects among available processes for next execution on CPU
- 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

Ready queue and various I/O device queues

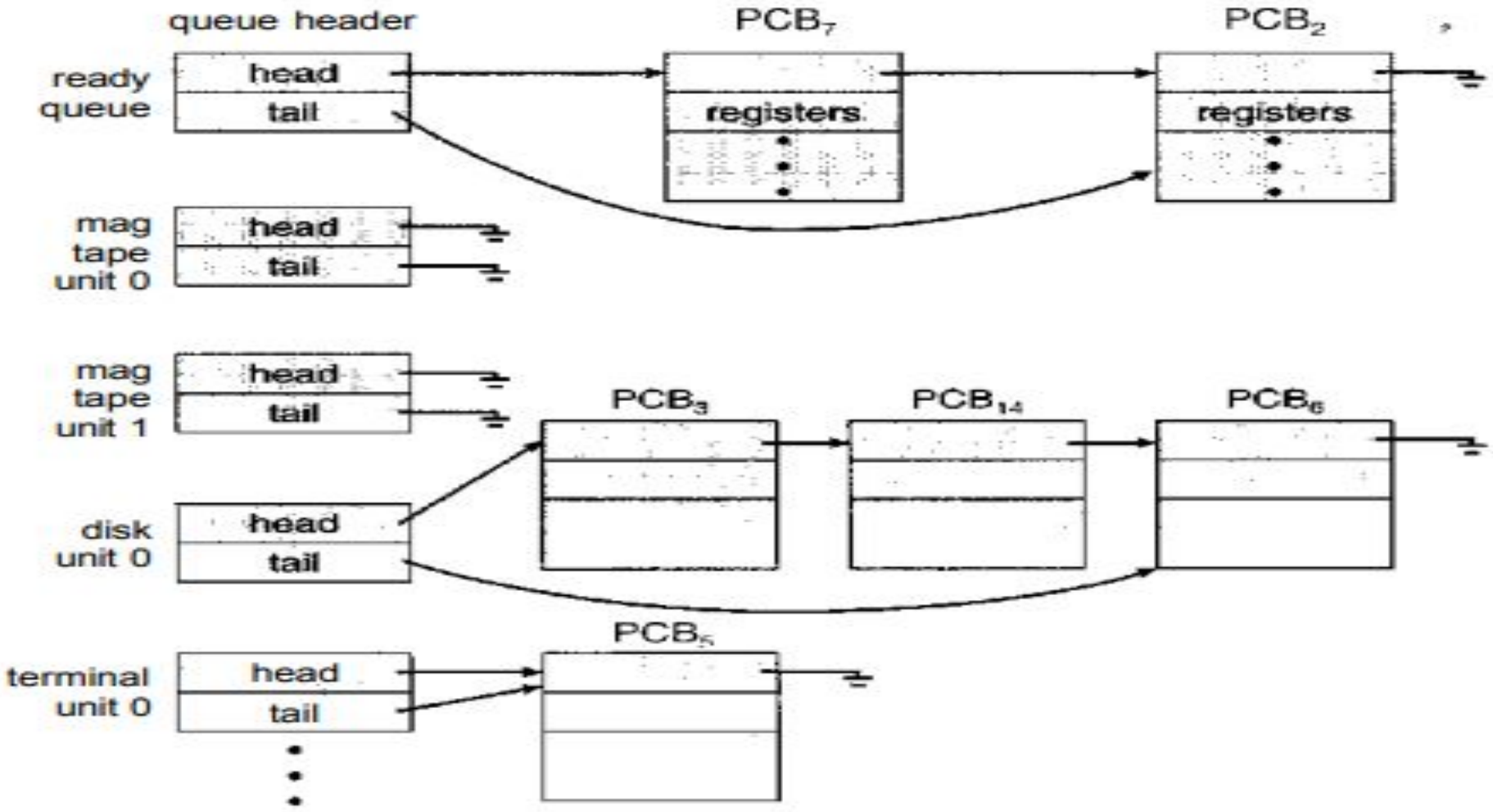
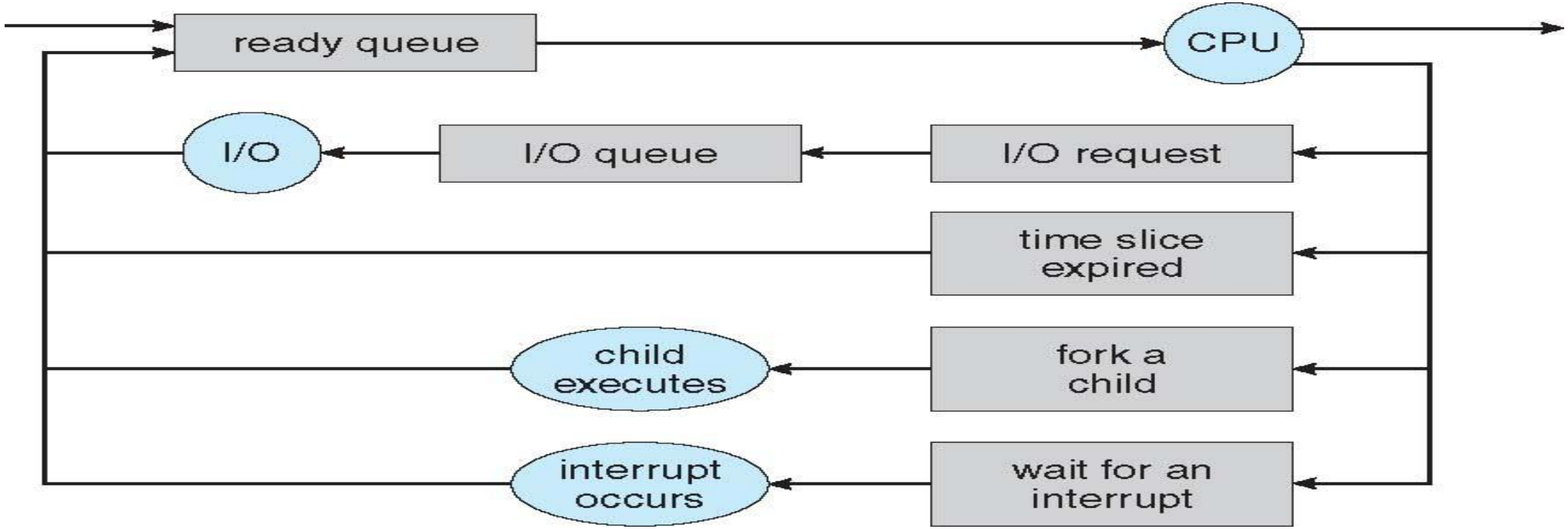


Figure 3.6 The ready queue and various I/O device queues.

Continued….

- A common representation of process scheduling is a **queuing diagram shown** in figure(next slide).
- Two types of queues are present: the ready queue and a set of device queues. The circles represent the resources that serve the queues, and the arrows indicate the flow of processes in the system.
- A new process is initially put in the ready queue. It waits there until it is selected for execution, or is dispatched. Once the process is allocated the CPU and is executing, one of several events could occur.
  - The process could issue an I/0 request and then be placed in an I/0 queue.
  - The process could create a new sub process and wait for the sub process's termination.
  - The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.



3.2.2 Schedulers

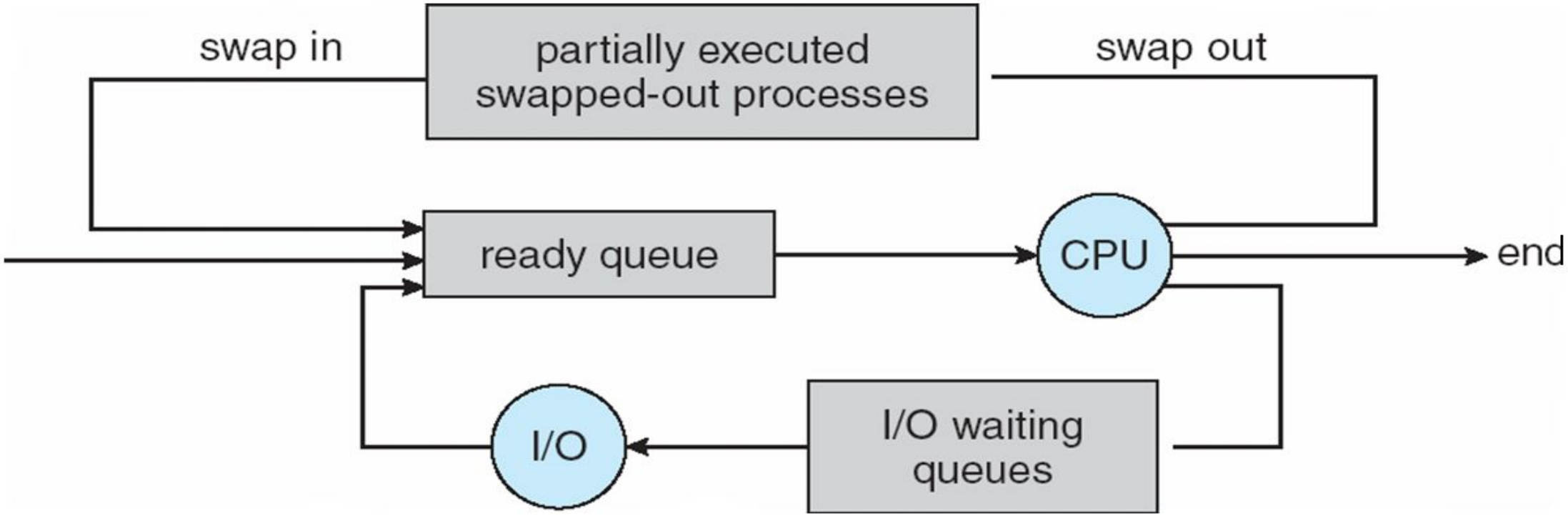
- The operating system must select, for scheduling purposes, processes from these queues in some fashion. The selection process is carried out by the appropriate scheduler.
- **Often, in a batch system, more processes are submitted than can be executed immediately. These processes are spooled to a mass-storage device (typically a disk), where they are kept for later execution. The long-term scheduler, or job scheduler, selects processes from this pool and loads them into memory for execution.**
- The short-term scheduler, or CPU scheduler, selects from among the processes that are ready to execute and allocates the CPU to one of them.
- The primary distinction between these two schedulers lies in frequency of execution. The short-term scheduler must select a new process for the CPU frequently.
- A process may execute for only a few milliseconds before waiting for an I/0 request. Often, the short-term scheduler executes at least once every 100 milliseconds. Because of the short time between executions, the short-term scheduler must be fast.
- The long-term scheduler executes much less frequently. It controls **degree of multiprogramming**.

Continued….

- **If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system**. Thus, the long-term scheduler may need to be invoked only when a process leaves the system.
- It is important that the long-term scheduler make a careful selection. In general, most processes can be described as either I/ 0 bound or CPU bound. An I/O-bound process is one that spends more of its time doing I/O than it spends doing computations. A CPU-bound process, in contrast, generates I/0 requests infrequently, using more of its time doing computations.
- **It is important that the long-term scheduler select a good process mix of I/O-bound and CPU-bound processes.**
- **Some operating systems, such as time-sharing systems, may introduce an additional, intermediate level of scheduling i,e medium term scheduler as shown in figure(next slide).**
- The key idea behind a medium-term scheduler is that sometimes it can be advantageous to remove processes from memory (and from active contention for the CPU) and thus reduce the degree of multiprogramming. Later, the process can be reintroduced into memory, and its execution can be continued where it left off. This scheme is called swapping. The process is swapped out, and is later swapped in, by the medium-term scheduler. Swapping may be necessary to improve the process mix or because a change in memory requirements has overcommitted available memory, requiring memory to be freed up.

Continued…

Addition of Medium Term Scheduling



3.2.3 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
- Time dependent on hardware support
  - Some hardware provides multiple sets of registers per CPU -> multiple contexts loaded at once

Process scheduling

5.1 basic concept

In a single-processor system, only one process can run at a time; any others must wait until the CPU is free and can be rescheduled. The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization. The idea is relatively simple. A process is executed until it must wait, typically for the completion of some I/O request. In a simple computer system, the CPU then just sits idle. All this waiting time is wasted; no useful work is accomplished. With multiprogramming, we try to use this time productively. Several processes are kept in memory at one time. When one process has to wait, the operating system takes the CPU away from that process and gives the CPU to another process.

5.1.1 CPU-i/O Burst Cycle

process execution consists of a cycle of CPU execution and I/0 wait. Processes alternate between these two states. Process execution begins with a CPU burst. That is followed by an I/O burst, which is followed by another CPU burst, then another I/0 burst, and so on. Eventually, the final CPU burst ends with a system request to terminate execution (Figure 5.1).

Continued..

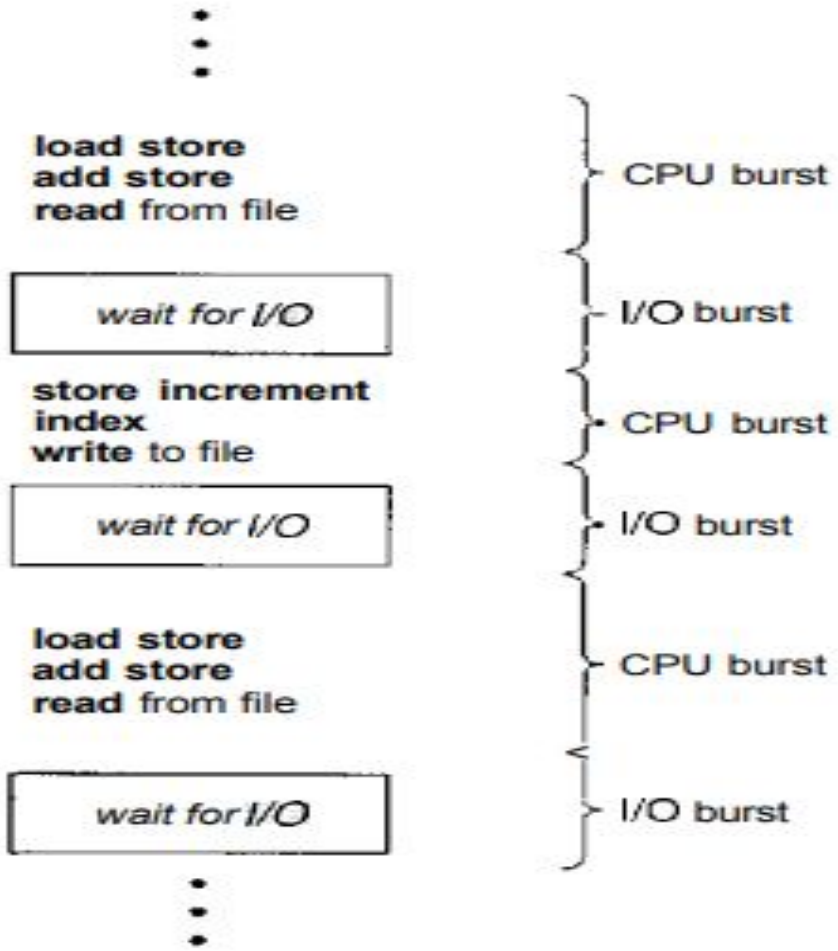


Figure 5.1 Alternating sequence of CPU and I/O bursts.

5.1.2 CPU Scheduler

- Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The selection process is carried out by the **short-term scheduler** (or CPU scheduler). The scheduler selects a process from the processes in memory that are ready to execute and allocates the CPU to that process.
- **Note that the ready queue is not necessarily a first-in, first-out (FIFO) queue**. As we shall see when we consider the various scheduling algorithms, **a ready queue can be implemented as a FIFO queue, a priority queue, a tree, or simply an unordered linked list.**

5.1.3 Preemptive Scheduling

CPU-scheduling decisions may take place under the following four circumstances:

1. When a process switches from the running state to the waiting state (for example, as the result of an I/0 request or an invocation of wait for the termination of one of the child processes)
2. When a process switches from the running state to the ready state (for example, when an interrupt occurs)

Continued.....

3. When a process switches from the waiting state to the ready state (for example, at completion of I/0)

4. When a process terminates.

- For situations 1 and 4, there is no choice in terms of scheduling. A new process (if one exists in the ready queue) must be selected for execution. There is a choice, however, for situations 2 and 3. **When scheduling takes place only under circumstances 1 and 4, we say that the scheduling scheme is nonpreemptive or cooperative**; **otherwise, it is preemptive**. Under nonpreemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state. This scheduling method was used by Microsoft Windows 3.x; Windows 95 introduced preemptive scheduling, and all subsequent versions of Windows operating systems have used preemptive scheduling. The Mac OS X operating system for the Macintosh also uses preemptive scheduling;

Continued…

5.1.4 Dispatcher

Another component involved in the CPU-scheduling function is the dispatcher. The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler. This function involves the following:

- Switching context
- Switching to user mode
- Jumping to the proper location in the user program to restart that program

The dispatcher should be as fast as possible, since it is invoked during every process switch. The time it takes for the dispatcher to stop one process and start another running is known as the dispatch latency.

Scheduling criteria

Different CPU-scheduling algorithms have different properties, and the choice of a particular algorithm may favor one class of processes over another. In choosing which algorithm to use in a particular situation, we must consider the properties of the various algorithms. The criteria include the following.

- **CPU utilization.** We want to keep the CPU as busy as possible. Conceptually, CPU utilization can range from 0 to 100 percent. In a real system, it should range from 40 percent (for a lightly loaded system) to 90 percent (for a heavily used system).
- **Throughput.** If the CPU is busy executing processes, then work is being done. One measure of work is the number of processes that are completed per time unit, called throughput. For long processes, this rate may be one process per hour; for short transactions, it may be ten processes per second.
- **Turnaround time.** From the point of view of a particular process, the important criterion is how long it takes to execute that process. **The interval from the time of submission of a process to the time of completion is the turnaround time. Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/0.**
- **Waiting time.** The CPU-scheduling algorithm does not affect the amount of time during which a process executes or does I/0; it affects only the amount of time that a process spends waiting in the ready queue. Waiting time is the sum of the periods spent waiting in the ready queue.
- **Response time.** In an interactive system, turnaround time may not be the best criterion. Often, a process can produce some output fairly early and can continue computing new results while previous results are being output to the user. **Thus, another measure is the time from the submission of a request until the first response is produced. This measure, called response time, is the time it takes to start responding, not the time it takes to output the response.** The turnaround time is generally limited by the speed of the output device.

Continued…

- **It is desirable to maximize CPU utilization and throughput and to minimize turnaround time, waiting time, and response time.** In most cases, we optimize the average measure. However, under some circumstances, it is desirable to optimize the minimum or maximum values rather than the average. For example, to guarantee that all users get good service, we may want to minimize the maximum response time.

5.3 scheduling algorithms

- CPU scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU. There are many different CPU-scheduling algorithms. In this section, we describe several of them.

- First –come , first served (FCFS)
- Shortest job first scheduling(SJF)
- Priority scheduling
- Round robin scheduling
- Multilevel queue scheduling
- Multilevel feedback queue scheduling.

Continued…

5.3.1 First-Come, First-Served Scheduling

- By far the simplest CPU-scheduling algorithm is the first-come, first-served (FCFS) scheduling algorithm. With this scheme, the process that requests the CPU first is allocated the CPU first. The implementation of the FCFS policy is easily managed with a FIFO queue. When a process enters the ready queue, its PCB is linked onto the tail of the queue. When the CPU is free, it is allocated to the process at the head of the queue. The running process is then removed from the queue.

Advantages of FCFS:

- The code for FCFS scheduling is simple to write and understand.

Disadvantages :

- On the negative side, the average waiting time under the FCFS policy is often quite long.
- There is **a convoy effect** if several other processes wait for the one big process to get off the CPU. This effect results in lower CPU and device utilization.

Continue…

consider the performance of FCFS scheduling in a dynamic situation. Assume we have one CPU-bound process and many I/O-bound processes. As the processes flow around the system, the following scenario may result. The CPU-bound process will get and hold the CPU. During this time, all the other processes will finish their I/0 and will move into the ready queue, waiting for the CPU. While the processes wait in the ready queue, the I/0 devices are idle. Eventually, the CPU-bound process finishes its CPU burst and moves to an I/0 device. All the I/O-bound processes, which have short CPU bursts, execute quickly and move back to the I/0 queues. At this point, the CPU sits idle. The CPU-bound process will then move back to the ready queue and be allocated the CPU. Again, all the I/0 processes end up waiting in the ready queue until the CPU-bound process is done. There is **a convoy effect** as all the other processes wait for the one big process to get off the CPU. This effect results in lower CPU and device utilization than might be possible if the shorter processes were allowed to go first.

- Note also that the FCFS scheduling algorithm is **nonpreemptive**. Once the CPU has been allocated to a process, that process keeps the CPU until it releases the CPU, either by terminating or by requesting I/0.

Problems :

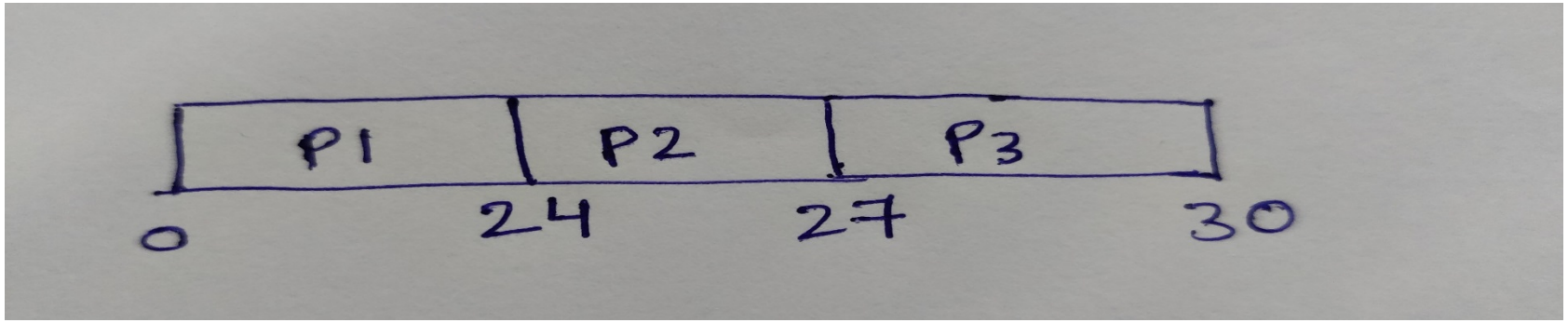
1.Problem statement

Example 1: draw a gantt chart and calculate average waiting time and average turnaround time for the following processes. Assume that all processes have arrived at time 0.

Processes	Burst time (ms)
p1	24
p2	3
p3	3

Solution.

Gantt chart



Processes	Burst time	Completion time(CT)	Turnaround time(TAT)	Waiting time(WT)
p1	24	24	24	0
p2	3	27	27	24
p3	3	30	30	27

Completion time: from gantt chart

Turnaround time(TAT): completion time - arrival time

Waiting time(WT): turnaround time - burst time

Continued..

Turn around time:
P1 24-0= 24,
P2 27-0= 27
P3 30-0= 30
Waiting time
P1 24-24=0
P2 27-3 = 24
P3 30-3 = 27
Average turnaround time= 24+27+30/3= 27 ms
Average waiting time= 0+24+27/3= 17ms

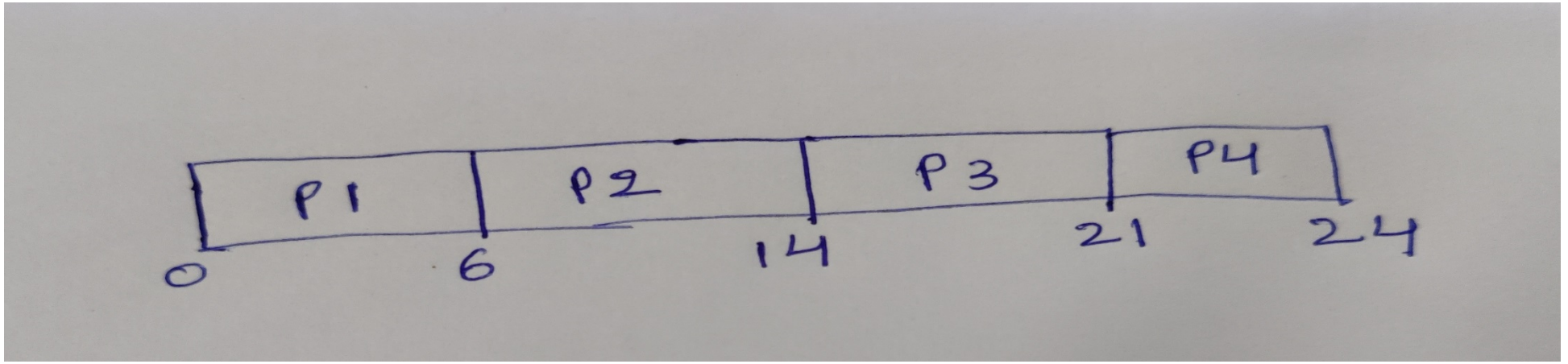
2. Problem statement

draw a gantt chart and calculate average waiting time and average turnaround time for the following processes. Assume that all processes have arrived at time 0.

Processes	Burst time (ms)
p1	6
p2	8
p3	7
p4	3

Solution.

Gantt chart



Processes	Burst time	Completion time(CT)	Turnaround time(TAT)	Waiting time(WT)
p1	6	6	6	0
p2	8	14	14	6
p3	7	21	21	14
p4	3	24	24	21

Completion time: from gantt chart

Turnaround time(TAT): completion time - arrival time avg turnaround time=16.25

Waiting time(WT): turnaround time - burst time avg waiting time=10.25

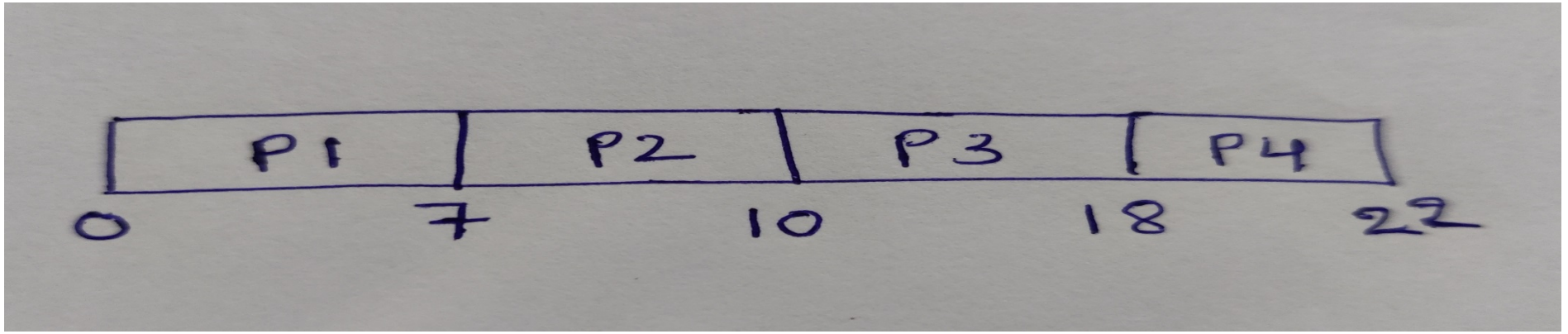
3. Problem statement

draw a gantt chart and calculate average waiting time and average turnaround time for the following processes using FCFS.

Processes	Burst time (ms)	Arrival time
p1	7	0
p2	3	2
p3	8	2
p4	4	3

Solution.

Gantt chart



Processes	Burst time	Arrival time	Completion time(CT)	Turnaround time(TAT)	Waiting time(WT)
p1	7	0	7	7	0
p2	3	2	10	8	5
p3	8	2	18	16	8
p4	4	3	22	19	15

Completion time: from gantt chart

Turnaround time(TAT): completion time - arrival time avg turnaround time= 12.5

Waiting time(WT): turnaround time - burst time avg waiting time= 7

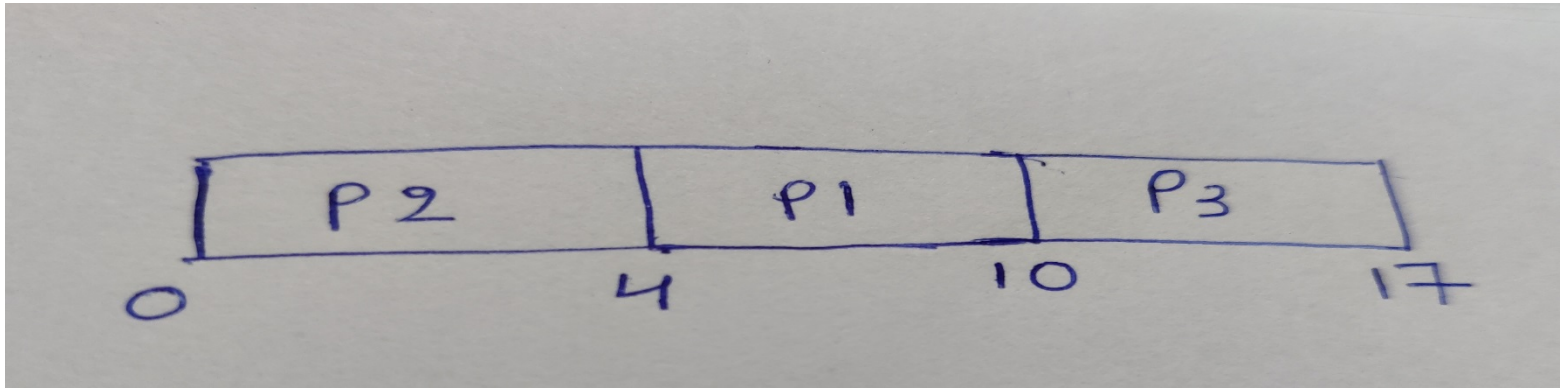
2. Problem statement

draw a gantt chart and calculate average waiting time and average turnaround time for the following processes using FCFS.

Processes	Arrival time	Burst time (ms)
p1	1	6
p2	0	4
p3	2	7

Solution.

Gantt chart



Processes	Burst time	Arrival time	Completion time(CT)	Turnaround time(TAT)	Waiting time(WT)
p1	6	1	10	9	3
p2	4	0	4	4	0
p3	7	2	17	15	8

Completion time: from gantt chart

Turnaround time(TAT): completion time - arrival time avg turnaround time= 9.3

Waiting time(WT): turnaround time - burst time avg waiting time= 3.6

5.3.2 Shortest-Job-First Scheduling

- A different approach to CPU scheduling is the shortest-job-first (SJF) scheduling algorithm. This algorithm associates with each process the length of the process's next CPU burst. When the CPU is available, it is assigned to the process that has the smallest next CPU burst. If the next CPU bursts of two processes are the same, FCFS scheduling is used to break the tie. Note that a more appropriate term for this scheduling method would be the shortest-next-CPU-burst algorithm, because scheduling depends on the length of the next CPU burst of a process, rather than its total length.

Advantage of SJF

- The SJF scheduling algorithm is provably optimal, in that it gives the minimum average waiting time for a given set of processes.

Disadvantage

- The real difficulty with the SJF algorithm is knowing the length of the next CPU request. For long-term (job) scheduling in a batch system, we can use as the length the process time limit that a user specifies when he submits the job. Thus, users are motivated to estimate the process time limit accurately, since a lower value may mean faster response.

Continued…

- **The SJF algorithm can be either preemptive or nonpreemptive**. The choice arises when a new process arrives at the ready queue while a previous process is still executing. The next CPU burst of the newly arrived process may be shorter than what is left of the currently executing process. A preemptive SJF algorithm will preempt the currently executing process, whereas a nonpreemptive SJF algorithm will allow the currently running process to finish its CPU burst. **Preemptive SJF scheduling is sometimes called shortest-remaining-time-first scheduling.**

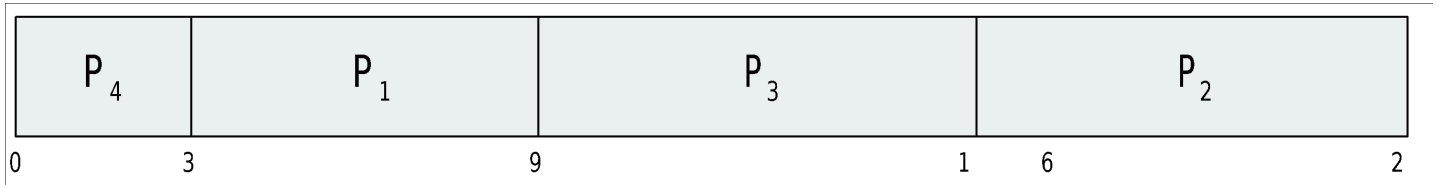
1. Problem statement

draw a gantt chart and calculate average waiting time and average turnaround time for the following processes using SJF scheduling algorithm. Assume that all processes have arrived at time 0.

Processes	CPU Burst time (ms)
p1	6
p2	8
p3	7
p4	3

Solution.

Gantt chart



Processes	Burst time	Completion time(CT)	Turnaround time(TAT)	Waiting time(WT)
p1	6	9	9	3
p2	8	24	24	16
p3	7	16	16	9
p4	3	3	3	0

Completion time: from gantt chart

Turnaround time(TAT): completion time - arrival time

Waiting time(WT): turnaround time - burst time

avg turnaround time=13ms

avg waiting time= 7ms

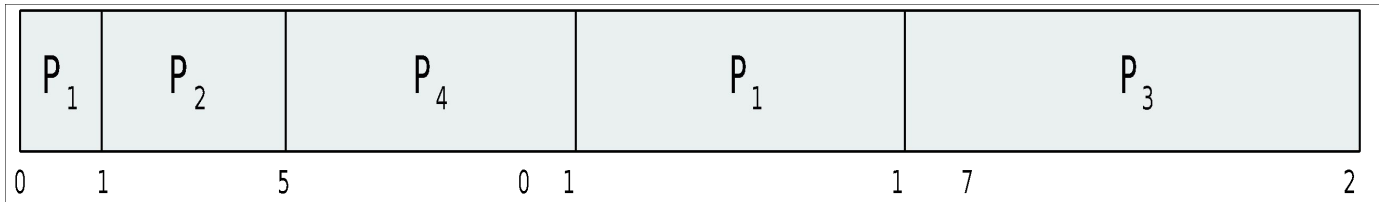
3. Problem statement

draw a gantt chart and calculate average waiting time and average turnaround time for the following processes using **preemptive SJF scheduling algorithm or shortest remaining time first scheduling.**

Processes	Arrival time	Burst time (ms)
p1	0	8
p2	1	4
p3	2	9
p4	3	5

Solution.

Gantt chart



Processes	Arrival time	Burst time	Completion time(CT)	Turnaround time(TAT)	Waiting time(WT)
p1	0	8	17	17	9
p2	1	4	5	4	0
p3	2	9	26	24	15
p4	3	5	10	7	2

Completion time: from gantt chart

Turnaround time(TAT): completion time - arrival time avg turanaround time=13 ms

Waiting time(WT): turnaround time - burst time avg waiting time= 6.5 ms

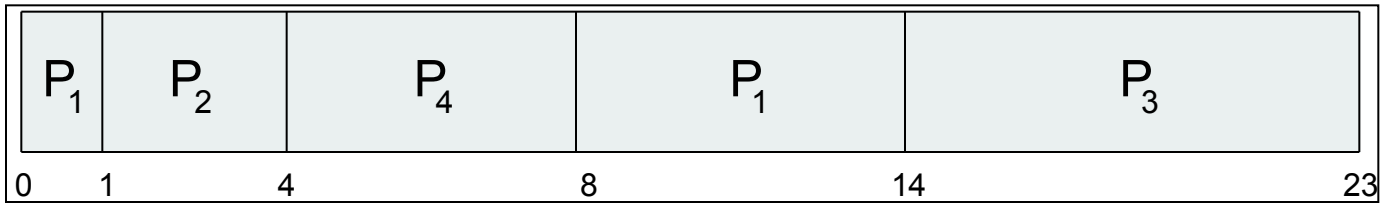
2. Problem statement

draw a gantt chart and calculate average waiting time and average turnaround time for the following processes using **preemptive SJF scheduling algorithm or shortest remaining time first scheduling.**

Processes	Arrival time	Burst time (ms)
p1	0	7
p2	1	3
p3	2	9
p4	3	4

Solution.

Gantt chart



Processes	Arrival time	Burst time	Completion time(CT)	Turnaround time(TAT)	Waiting time(WT)
p1	0	7	14	14	7
p2	1	3	4	3	0
p3	2	9	23	21	12
p4	3	4	8	5	1

Completion time: from gantt chart

Turnaround time(TAT): completion time - arrival time avg turn around time= 10.75

Waiting time(WT): turnaround time - burst time avg waiting time= 5

5.3.3 Priority Scheduling

- The SJF algorithm is a special case of the general priority scheduling algorithm. A priority is associated with each process, and the CPU is allocated to the process with the highest priority. Equal-priority processes are scheduled in FCFS order. **An SJF algorithm is simply a priority algorithm where the priority (p) is the inverse of the (predicted) next CPU burst**. The larger the CPU burst, the lower the priority, and vice versa.
- Note that we discuss scheduling in terms of high priority and low priority. Priorities are generally indicated by some fixed range of numbers, such as 0 to 7 or 0 to 4,095. However, there is no general agreement on whether 0 is the highest or lowest priority. Some systems use low numbers to represent low priority; others use low numbers for high priority. This difference can lead to confusion. In this text, we assume that low numbers represent high priority.

Continued…

- **Priority scheduling can be either preemptive or nonpreemptive. When a process arrives at the ready queue, its priority is compared with the priority of the currently running process. A preemptive priority scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority of the currently running process. A nonpreemptive priority scheduling algorithm will simply put the new process at the head of the ready queue.**

- A major problem with priority scheduling algorithms is indefinite blocking, or starvation. A process that is ready to run but waiting for the CPU can be considered blocked. A priority scheduling algorithm can leave some low priority processes waiting indefinitely.
- **A solution to the problem of indefinite blockage of low-priority processes is aging. Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time.**

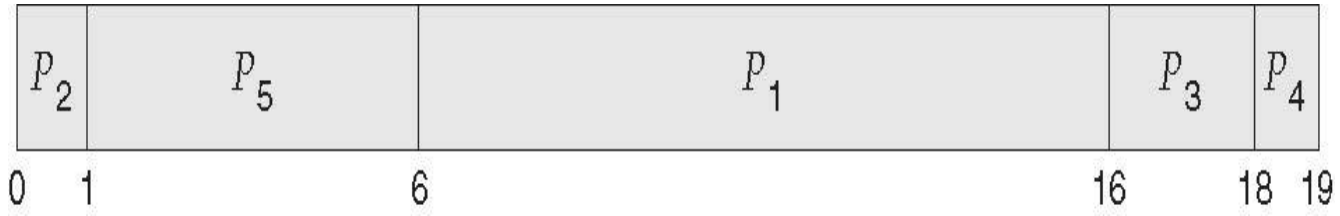
1. Problem statement

draw a gantt chart and calculate average waiting time and average turnaround time for the following processes using priority scheduling. Assume arrival time is 0 and lower number indicates higher priority.

Processes	Burst time(ms)	priority
p1	10	3
p2	1	1
p3	2	4
p4	1	5
p5	5	2

Solution.

Gantt chart



Processes	Burst time	priority	Completion time(CT)	Turnaround time(TAT)	Waiting time(WT)
p1	10	3	16	16	6
p2	1	1	1	1	0
p3	2	4	18	18	16
p4	1	5	19	19	18
p5	5	2	6	6	1

Completion time: from gantt chart

Turnaround time(TAT): completion time - arrival time avg turnaround time=12ms

Waiting time(WT): turnaround time - burst time avg waiting time= 8.2 ms

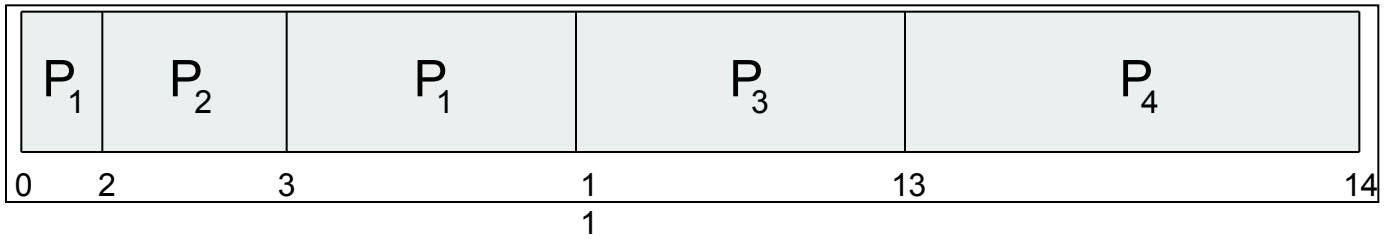
1. Problem statement

draw a gantt chart and calculate average waiting time and average turnaround time for the following processes using **preemptive priority scheduling** . Lower number indicates higher priority.

Processes	Arrival time	Burst time(ms)	priority
p1	0	10	2
p2	2	1	1
p3	3	2	3
p4	4	1	4

Solution.

Gantt chart



Processes	Arrival time	Burst time	priority	Completion time(CT)	Turnaround time(TAT)	Waiting time(WT)
p1	0	10	2	11	11	1
p2	2	1	1	3	1	0
p3	3	2	3	13	10	8
p4	4	1	4	14	10	9

Completion time: from gantt chart

Turnaround time(TAT): completion time - arrival time avg turnaround time= 8

Waiting time(WT): turnaround time - burst time avg waiting time= 4.5

5.3.4 Round-Robin Scheduling

- The round-robin (RR) scheduling algorithm is designed especially for timesharing systems. It is similar to FCFS scheduling, but preemption is added to enable the system to switch between processes. A small unit of time, called a time quantum or time slice, is defined. A time quantum is generally fronc 10 to 100 milliseconds in length.
- The ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum.
- One of two things will then happen. The process may have a CPU burst of less than 1 time quantum. In this case, the process itself will release the CPU voluntarily. The scheduler will then proceed to the next process in the ready queue. Otherwise, if the CPU burst of the currently running process is longer than 1 time quantum, the timer will go off and will cause an interrupt to the operating system.

Continued…

- The performance of the RR algorithm depends heavily on the size of the time quantum. At one extreme, if the time quantum is extremely large, the RR policy is the same as the FCFS policy. In contrast, if the time quantum is extremely small (say, 1 millisecond), the RR approach is called processor sharing and (in theory) creates the appearance that each of 11 processes has its own processor running at 1 I 11 the speed of the real processor. This approach was used in Control Data Corporation (CDC) hardware to implement ten peripheral processors with only one set of hardware and ten sets of registers.
- we want the time quantum to be large with respect to the context switch time. If the context-switch time is approximately 10 percent of the time quantum, then about 10 percent of the CPU time will be spent in context switching. In practice, most modern systems have time quanta ranging from 10 to 100 milliseconds. The time required for a context switch is typically less than 10 microseconds; thus, the context-switch time is a small fraction of the time quantum.

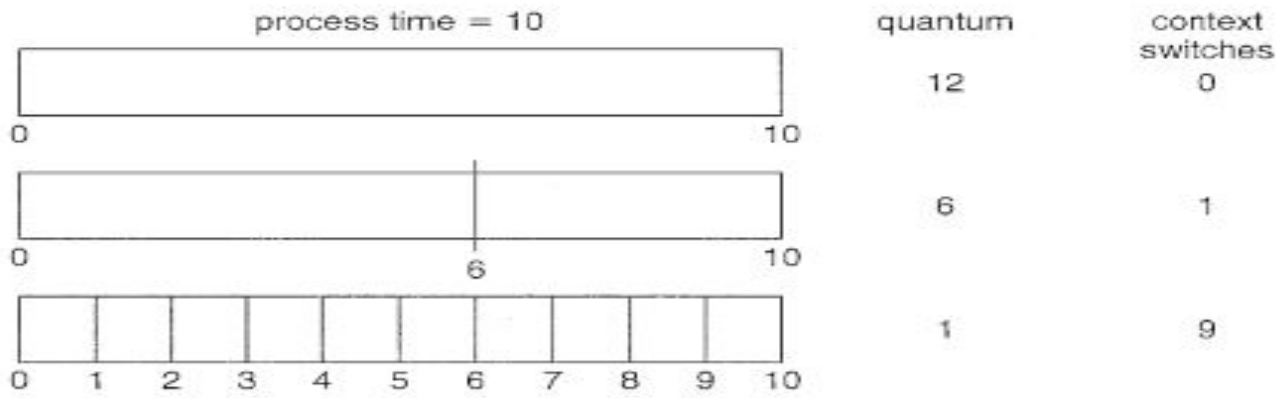


Figure 5.4 How a smaller time quantum increases context switches.

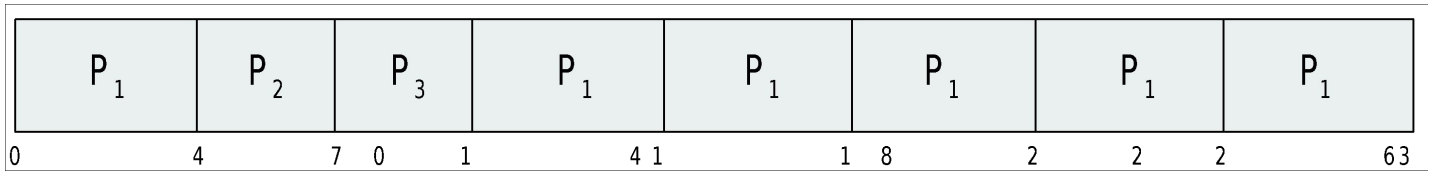
1. Problem statement

draw a gantt chart and calculate average waiting time and average turnaround time for the following processes using round robin scheduling algorithm. The time quantum is 4 ms.

Processes	Burst time(ms)
p1	24
p2	3
p3	3

Solution.

Gantt chart



Processes	Burst time	Completion time(CT)	Turnaround time(TAT)	Waiting time(WT)
p1	24	30	30	6
p2	3	7	7	4
p3	3	10	10	7

Completion time: from gantt chart

Turnaround time(TAT): completion time - arrival time avg turnaround time=15.6ms

Waiting time(WT): turnaround time - burst time avg waiting time=5.6 ms

5.3.5 Multilevel Queue Scheduling

- Another class of scheduling algorithms has been created for situations in which processes are easily classified into different groups. For example, a common division is made between foreground (interactive) processes and background (batch) processes. These two types of processes have different response-time requirements and so may have different scheduling needs. In addition, foreground processes may have priority (externally defined) over background processes.
- A multilevel queue scheduling algorithm partitions the ready queue into several separate queues (Figure 5.6). The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type. Each queue has its own scheduling algorithm. For example, separate queues might be used for foreground and background processes. The foreground queue might be scheduled by an RR algorithm, while the background queue is scheduled by an FCFS algorithm.
- In addition, there must be scheduling among the queues, which is commonly implemented as fixed-priority preemptive scheduling. For example, the foreground queue may have absolute priority over the background queue.

Continued…

- Let's look at an example of a multilevel queue scheduling algorithm with five queues, listed below in order of priority:
- System processes
- Interactive processes
- Interactive editing processes
- Batch processes
- Student processes

Each queue has absolute priority over lower-priority queues. No process in the batch queue, for example, could run unless the queues for system processes, interactive processes, and interactive editing processes were all empty. If an interactive editing process entered the ready queue while a batch process was running, the batch process would be preempted.

5.3.6 Multilevel Feedback Queue Scheduling

- Normally, when the multilevel queue scheduling algorithm is used, processes are permanently assigned to a queue when they enter the system. If there are separate queues for foreground and background processes, for example, processes do not move from one queue to the other, since processes do not change their foreground or background nature. This setup has the advantage of low scheduling overhead, but it is inflexible.
- The multilevel feedback queue scheduling algorithm, in contrast, allows a process to move between queues. The idea is to separate processes according to the characteristics of their CPU bursts. If a process uses too much CPU time, it will be moved to a lower-priority queue. This scheme leaves I/O-bound and interactive processes in the higher-priority queues. In addition, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue. This form of aging prevents starvation.
- For example, consider a multilevel feedback queue scheduler with three queues, numbered from 0 to 2 (Figure 5.7). The scheduler first executes all processes in queue 0. Only when queue 0 is empty will it execute processes in queue 1. Similarly, processes in queue 2 will only be executed if queues 0 and 1 are empty. A process that arrives for queue 1 will preempt a process in queue 2. A process in queue 1 will in turn be preempted by a process arriving for queue 0.

Continued…

- A process entering the ready queue is put in queue 0. A process in queue 0 is given a time quantum of 8 milliseconds. If it does not finish within this time, it is moved to the tail of queue 1. If queue 0 is empty, the process at the head of queue 1 is given a quantum of 16 milliseconds. If it does not complete, it is preempted and is put into queue 2. Processes in queue 2 are run on an FCFS basis but are run only when queues 0 and 1 are empty.

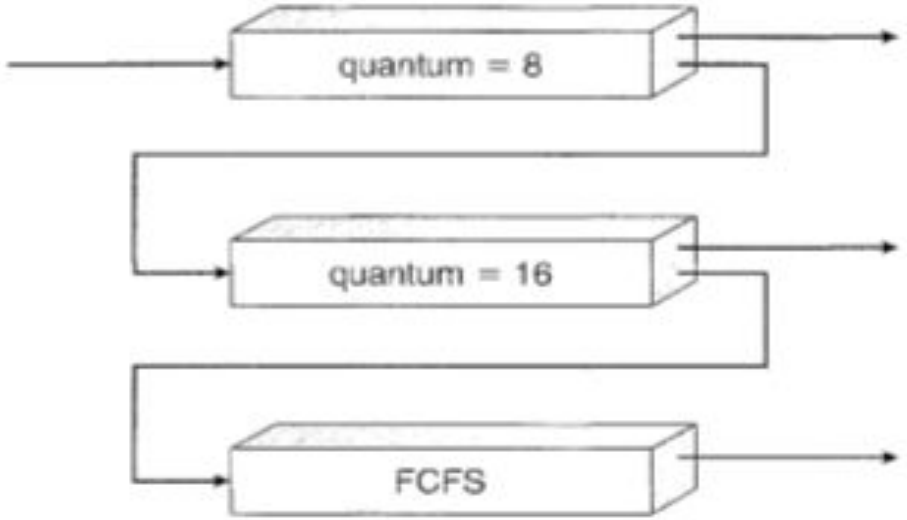


Figure 5.7 Multilevel feedback queues.

Continued…

In general, a multilevel feedback queue scheduler is defined by the following parameters:

- The number of queues
- The scheduling algorithm for each queue
- The method used to determine when to upgrade a process to a higher priority queue
- The method used to determine when to demote a process to a lower priority queue
- The method used to determine which queue a process will enter when that process needs service