Unit 3 : Process Management

Processes are the most widely used units of computation in programming and systems, although object and threads are becoming more prominent in contemporary systems. Process management and CPU scheduling is the basis of multiprogramming operating system. By switching the CPU between processes the operating system can make the computer more productive. In this unit we will discuss the details of process and scheduling concepts. The concepts of semaphores, process synchronization, mutual exclusion and concurrency are all central to the study of operating systems and to the field of computing in general. The lessons 6 and 7 focuses on these. The lesson 8 of this unit focuses on IPC through message passing and classical IPC problem.

Lesson 1 : Process Concept

1.1. Learning Objectives

On completion of this lesson you will know :

- multiprogramming
- process and process state
- PCM.

1.2. Multiprogramming and its Problem

Multiprogramming is where multiple processes are in the process of being executed.

Multiprogramming is where multiple processes are in the process of being executed. Only one process is ever actually running on the CPU. The remaining process are in a number of others states including.

blocked,

Waiting for some event to occur, this includes some form of I/O to complete.

ready,

Able to be executed just waiting on the ready queue for its turn on the CPU.

The important part of multiprogramming is the execution is interleaved with I/O. This makes it possible for one process to be executing on the CPU and for other processes to be performing some form of I/O operator. This provides more efficient use of all of the resources available to the operating system.

Problems involved with multiprogramming are described below

resource management

Multiple processes must share a limited number of resources including the CPU, memory, I/O devices etc. These resources must be allocated in a safe, efficient and fair manner. This can be difficult and provides more overhead.

Problems

protection
Processes must be prevented from accessing each others resources.

mutual exclusion and critical sections
there are times when a process must be assured that it is the only process using some resource

extra overhead for the operating system
The OS is responsible for performing all of these tasks. These tasks require additional code to be written and then executed.

Benefits

Benefits

The benefits of multiprogramming are increased CPU utilization and higher total job throughput. Throughput is the amount of work accomplished in a given time interval.

1.3. Process

A process is defined as an instance of a program in execution. A process is a sequential unit of computation. The action of the unit of computation is described by a set of instructions executed sequentially on a Von-Neuman computer, using a set of data associated with the process. The components of a process are the programs to be excepted. The data on which the program will execute, resources required by the program (e.g. memory) and the status of the execution. For the process to execute; it must have a suitable abstract machine environment. *In various operating systems, processes may be called jobs, users, programs, tasks or activities. A process may be considered as a job or time shared program.*

A process is defined as an instances of a program in execution.

1.3.1. Process State

As the program executes, the process changes state. The state of a process is defined by its current activity. Process execution is an alternating sequence of CPU and I/O bursts; beginning and ending with a CPU burst. Thus each process may in one of the following states; *new, ready, running, blocked* or *halted*. A process which is waiting for the CPU is *ready*. A process which has been allocated the CPU is *running*. A process which is wanting for some thing to occur is *blocked*.

The state of a process is defined by its current activity.

Some systems may add a suspended state. Processes will become suspended when the operating system decides that too many processes are on the system. In an attempt to lighten the load the processes will be suspended. Most of their resources will be released.

At some later time when the system load has been reduced the operating system may change the processes state back.

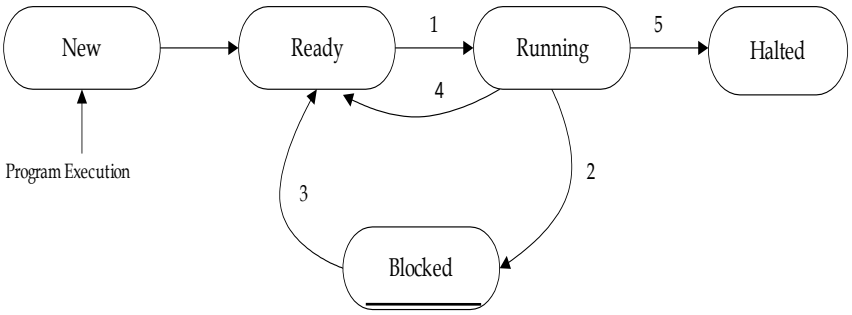


Fig. 3.1 : Process state diagram.

Reasons for Transition

1. Operating system places process onto CPU.
2. Process has to wait for some event to occur.
   a) Some form of I/O.
   b) an interrupt must be handled.
3. The event of the process was waiting on has occurred.
4. The processes quantum has expired.
5. The process finishes execution.

1.3.2. CPU-I/O Burst Cycle

CPU-I/O Burst Cycle

The success of CPU scheduling depends upon the following observed property of processes : ***CPU burst*** and ***I/O burst.*** Process execution is a cycle of CPU execution and I/O wait. Processes alternate back and forth between these two states. Process execution begins with a CPU burst. It is followed by an I/O burst, which is followed by another CPU burst, then another I/O burst, and so on. Eventually, the last CPU burst will end with a system request to terminate execution, rather than another I/O burst (Fig. 3.2).

```
LOAD
STORE                    CPU burst
SUB
STORE
LOAD from file
     wait for I/O        I/O burst

STORE
INCREMENT INDEX          CPU burst
WRITE to file

                         I/O burst
     wait for I/O
```

Fig. 3.2 : Sequence of CPU and I/O bursts.

The duration's of these CPU bursts have been measured. Although they vary greatly from process to process and computer to computer. There are a very large number of very short CPU bursts and a small number of very long ones. An I/O-bound program might have a few very long CPU bursts. It is important to select appropriate CPU scheduling algorithm.

1.3.3. Process Control Block

A process control block (PCB) is a data block or record containing many pieces of the information associated with a specific process.

The operating system groups all information that it needs about a particular process into a data structure called a process descriptor or a process control block (PCB). Whenever a process is created (initialized, installed), the operating system creates a corresponding process control block to serve as its run-time description during the lifetime of the process. When the process terminates, its PCB is released to the pool of free cells from which new PCBs are drawn. A Process becomes known to the operating system and thus eligible to compete for system resources only when it has an active PCB associated with it.

Each process is represented in the operating system by its own process control block (also called a tack control block or a job control block). A process control block (PCB) is a data block or record containing many pieces of the information associated with a specific process. The information stored in PCB typically includes.

- Process ID /Number
- The process state may be new ready, running, or halted.
- The program counter indicates the address of the next instruction to be executed for this process.
- The CPU registers
- Any memory management information, including base and bounds registers or page tables.
- Any accounting information, including the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.
- I/O status information
- CPU scheduling information, including a process priority, pointers to scheduling queues, and any other scheduling parameters.

The PCB simply serves as the repository for any information that may vary from process to process.

1.4. Exercise

1.4.1. Multiple choice questions

1. Which of the following process state transition is invalid?

i) ready → running
ii) ready → blocked
iii) blocked → ready
iv) running → dead.

1.4.2. Questions for short answers

1. Multiprogramming

i) Why it is used?
ii) What problems does using it produce?
iii) What benefits does using it produce?

2. Describe the life cycle of a process?
3. What possible states are involved?
4. What are reasons for a transition from one states to another?

1.4.3. Analytical questions

a) What do you understand by PCB? Briefly describe.
b) What are the components of a processor?

Lesson 2 : Scheduling Concept

2.1. Learning Objectives

On completion of the lesson you will be able to know :

- scheduling and scheduling queues
- what a scheduler is
- different types of schedulers.

Scheduling is a fundamental operating system function, since almost all computer resources are scheduled before use. The CPU is of course, one of the primary computer resources. Thus its scheduling is central to operating system design.

When more than one process is run-able, the OS must decide which one to run first. That part of the OS concerned with this decision is called scheduler and the algorithm it uses is called scheduling algorithm.

Scheduling

Scheduling refers to a set of policies and mechanism into operating system that govern the order in which the work to be done by a computer system is complicated.

Scheduler

Scheduler is an OS module that selects the next job to be admitted into the system and the next process to run. The primary objective of scheduling is to optimize system performance in accordance with the criteria deemed most important by the system designer. Before discussing about schedulers, we have to know about scheduling queues. Let's look at scheduling queues.

2.2.1. Scheduling Queues

The ready queues is used to contain all processes that are ready to be placed on to the CPU. The processes which are ready and waiting to execute are kept on a list called the ready queue. This list is generally a linked list. A ready queue header will contain pointers to the first and last PCBs in the list. Each PCB has a pointer field which points to the next process in the ready queue.

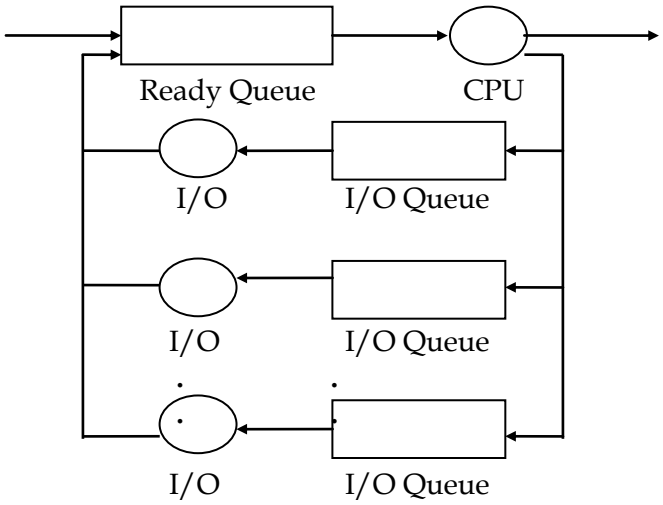


Fig. 3.3 : The ready queue.

The ready queue is not necessarily a first-in-first-out (FIFO) queue. A ready queue may be implemented as a FIFO queue, a priority queue, a tree, a stack, or simply an unordered list. Conceptually, however, all of the processes in the ready are lined up waiting for a chance to run on the CPU.

The list of processes waiting for a particular I/O device is called a device queue. Each device has its own device queue. If the device is a dedicated device, the device queue will never have more than one process in it. If the device is sharable, several processes may be in the device queue.

A common representation for a discussion of CPU scheduling is a queuing diagram such as Fig. 3.3. Each rectangular box represents a queue. Two types of queues are present : the *ready queue* and a set of *device queues*. The circles represent the resources which serve the queues, and the arrows indicate the flow of processes in the system.

A process enters the system from the outside world and is put in the ready queue. It waits in the ready queue until it is selected for the CPU. After running on the CPU, it waits for an I/O operation by moving to an I/O queue. Eventually, it is served by the I/O device and returns to the ready queue. A process continue this CPU-I/O cycle until it finishes; then it exits from the system.

Since our main concern at this time is CPU scheduling, we can replace with one I/O waiting queue and I/O server.

2.2.2. Types of Schedulers

Types of Schedulers

In general, there are three different types of schedulers, which may coexist in a complex operating system : long-term, medium-term, and short-term schedulers.

Fig. 3.4 shows the possible traversal paths of jobs and programs through the components and queues, depicted by rectangles, of a computer system. The primary places of action of the three types of schedulers are marked with down-arrows. As shown in Fig. 3.4, a submitted batch job joins the batch queue while waiting to be processed by the long-term scheduler. Once scheduled for execution, processes spawned by the job enter the ready queue to await processor allocation by the short-term scheduler. After becoming suspended, the running process may be removed from memory and swapped out to secondary storage. Such processes are subsequently admitted to main memory by the medium-term scheduler in order to be considered for execution by the short-term scheduler.

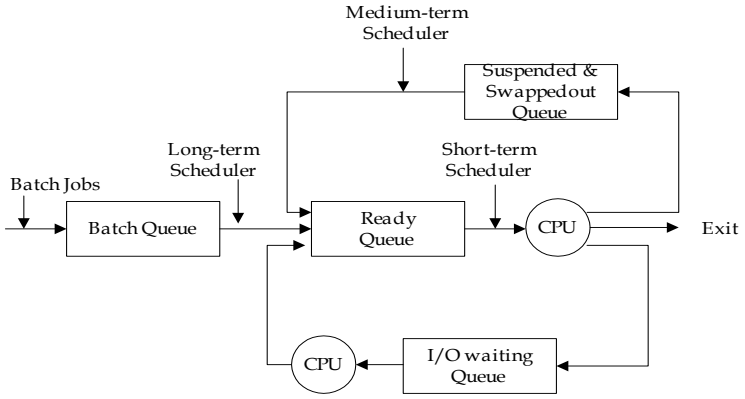


Fig. 3.4 : Schedulers.

Long-term scheduler

Long Term scheduling : The decision to add to the pool of processes to be executed.

The long-term scheduler (or job scheduler) works with the batch queue and determines which jobs are admitted to the system for processing. In a batch system, there are often more jobs submitted than can be executed immediately. These jobs are spooled to a mass storage device (typically a disk), where they are kept for later execution. The long-term scheduler selects jobs from this job pool and loads them into memory for execution.

The primary objective of the long-term scheduler is to provide a balanced mix of jobs, such as CPU bound and I/O bound, to the short-term scheduler. In a way, the long-term scheduler acts as a first-level throttle in keeping resource utilization of the desired level.

Short-term scheduler

Short term scheduling : The decision as to which process will gain the processor.

The short-term scheduler (or CPU scheduler) selects from among the jobs in memory which are ready to execute and allocates the CPU to one of them.

Distinction between long-term and short-term schedulers

The primary distinction between these two schedulers is the frequency of their execution. The short-term scheduler must select a new process for the CPU quite often. A process may execute only a few milliseconds before waiting for an I/O request. Often the short-term scheduler executes at least once every 10 milliseconds. Because of the short duration between executions, the short-term scheduler must be very fast.

The long-term scheduler, on the other hand, executes much less frequently. It may be minutes between the arrival of new jobs in the system. The long-term scheduler controls the degree of multiprogramming (the number of processes in memory).

Medium-term scheduler

Medium term scheduling : The decision to add to the process in main memory.

The key idea behind a medium-term scheduler is that it can sometimes be advantageous to remove processes from memory (and from active contention for the CPU) and thus reduce the degree of multiprogramming. At some later time, the process can be reintroduced into memory and continued where it left off. This scheme is often called swapping. The process is swapped out and swapped in later by the medium-term scheduler. Swapping may be necessary to improve the job mix, or because a change in memory requirements has over committed available memory, requiring memory to be freed up.

Dispatcher

Another component involved in the CPU scheduler function is the dispatcher. The dispatcher is the module that actually gives control of the CPU to the process selected by the short-term scheduler. This function involves loading the registers of the process, switching to user mode, and jumping to the proper location in the user program to restart it. Obviously, the dispatcher should be as fast as possible.

2.3. Exercise

2.3.1. Multiple choice questions

1. The primary distinction between short term and long term scheduler is

i) the frequency of their execution.
ii) time
iii) speed
iv) number of throughput.

2.3.2. Questions for short answers

a) How many types of scheduling queues are there in this lesson? Describe briefly.
b) What do you understand by scheduling and scheduler?
c) What do you know about dispatcher?

2.3.3. Analytical questions

a) What do you understand by schedulers? Describe different type of schedulers.
b) Distinguish between short term, medium term and long term scheduling.

Lesson 3 : Scheduling Criteria and Algorithms

3.1. Learning Objectives

On completion of this lesson you will know :

- scheduling criteria of algorithms
- the criteria and aim
- important factors of scheduling mechanism
- preemptive and non-preemptive scheduling
- FCFS and shortest job first algorithms.

3.2. Scheduling Criteria

Scheduling Criteria

CPU scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU.

Before we look at algorithms for scheduling the CPU, we must first look at the criteria which might make a "good' scheduling algorithm. Many of the following criteria are contradictory. The criteria fall into the following categories.

- user-oriented to system-oriented criteria
- performance or there criteria.

User- Oriented, Performance Criteria

Criteria	**Aim**
Response Time (the amount of time it takes to start responding).	low response time, maximum number of interactive users.
Turnaround Time (The time between submission & completion)	minimize turnaround time.
Deadlines	maximize deadlines met.
Waiting time (the amount of time a job spends waiting in the ready queue).	low waiting time.

User-Oriented, Other Criteria

Criteria	**Aim**
Predictability	same cost and time usage no matter when a process is run.

System-Oriented, Performance Criteria

Criteria	Aim
Throughput (the number of jobs which are completed per time unit).	allow maximum number of jobs of complete.
Processor Utilization	maximize percentage of time, keep processor as busy as possible.
Overhead	minimize time processor busy executing OS.

System Oriented, Other Criteria

Criteria	Aim
Fairness	treat processes the same avoid starvation.
Enforcing Priorities	give preference to higher priority processes.
Balancing Resources	keep the system resources busy.

Important Factors

Factors which affect a scheduling mechanism are :

- I/O boundedness of a process
- CPU boundedness of a process
- is the process interactive or batch
- process priority
- page fault frequency
- preemption frequency
- execution time received
- execution time required to complete.

3.3. Non-preemptive VS Preemptive Scheduling

In general, scheduling algorithm may preemptive or non preemptive.

Preemptive scheduling is useful in a situation where you have high priority processes.

Non-preemptive VS Preemptive Scheduling

In batch, non-preemption implies that, once scheduled, a selected job runs to completion. With short-term scheduling, on preemption implies that the running process retains ownership of allocated resources, including the processor, until it voluntarily surrenders control to the operating system. In other words, the running process is not forced to relinquish ownership of the processor when a higher-priority process becomes ready for execution. However, when the running process

becomes suspended as a result of its own action, say, by waiting for an I/O completion, another ready process may be scheduled.

With preemptive scheduling a running process may be replaced by a higher-priority process at any time. This is accomplished by activating the scheduler whenever an event that changes the state of the system is detected. Since such preemption generally necessitates more frequent execution of the scheduler.

3.4. Scheduling Algorithm

There are different types of scheduling algorithm. In this lesson, we will describe only about FCFS and SJF, the rest of the algorithm will be discussed in the following lesson.

3.4.1. First-come First Served (FCFS)

First-Come First-Served (FCFS) is by far the simplest CPU scheduling algorithm. The work load is simply processed in the order of arrival, with no preemption. The process which request the CPU first is allocated the CPU first.

Implementation

The implementation of FCFS is easily managed with a FIFO (First in-First out) queue. As the process becomes ready, it joins the ready queue when the current process finishes, the oldest process is selected next.

Characteristics

- Simple to implement
- Non-preemptive
- Penalize short and I/O bound process.

Example

Considering the following example, FCFS will be clear to learners.

Job	Arrival Time	Service time
1	1	8
2	2	2
3	3	1
4	4	2
5	5	5

If the jobs arrive in the above order and are served in FIFS order, we get the result shown in the following Gantt chart.

Job 1	Job 2	Job 3	Job 4	Job 5	
1	9	11	12	14	19

From the Chart we can get the following Table :

Job	Time Arrive	Time Finish	Time Waiting	Turnaround Time
1	1	9	0	8
2	2	11	7	9
3	3	12	8	9
4	4	14	8	10
5	5	19	9	14

For easy calculation, we have to know the following term :

Turnaround time = Time _ Finish - time _arrive.

Wait time = Starting execution of the respective job- arriving time of the respective job.

So, the average turnaround time of process

$$= \frac{(8+9+9+10+14)}{5} = \frac{50}{5} = 10$$

Average wait time = $\frac{0+7+8+8+9}{5}$ = 32/5 = 6.4.

There is a convoy effect as all other processes wait for one big process to get off the CPU. This effect results in lower CPU and device utilization than might be possible if the shorter jobs allowed to go first.

Performance

FCFS scheduling may result in poor performance. As a consequence of no preemption, component utilization and the system throughput rate may be quite low.

3.4.2. Shortest-Job-First

A different approach to CPU scheduling is the Shortest-Job-First (SJF) algorithm. Shortest-Job-First associates with each job the length of its next service time. If two jobs have the same next CPU burst, FCFS is used. The process with the shortest expected execution time is given priority on the processor.

Characteristics

- Non-preemptive
- Reduces average waiting time over FIFO
- Must know how long a process will run
- Possible user abuse.

Example
As an example consider the following set of jobs.

Job	Arrival Time	Service Time
1	1	8
2	2	2
3	3	1
4	4	2
5	5	5

Using shortest-Job-First scheduling, we would schedule these jobs according to the following Gantt chart.

Job 1	Job 3	Job2	Job 4	Job 5	
1	9	10	12	14	19

From the Chart we can get the following Table :

Job	Time Arrive	Time Finish	Time Waiting	Turnaround Time
1	1	9	0	8 (9-1)
2	2	12	8 (10-2)	10 (12-2)
3	3	10	6 (9-3)	7 (10-3)
4	4	14	8 (12-4)	10 (14-4)
5	5	19	9 (14-5)	14 (19-5)

i) Average turnaround time = $\frac{8+10+7+10+14}{5} = 9.8$

ii) Average wait time = (0+8+6+8+9) /5 = 6..2

Shortest Job first is probably optimal, in that it gives the minimum average waiting time for a given set of jobs. The proof in Fig. 3.5 shows that moving a short job before a long one decreases the waiting time of the short job more than it increases the waiting time of the long job. So, the average waiting time decreases.

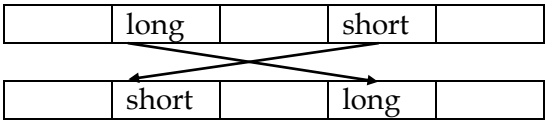


Fig.3.5 : Proving the SJF is optimal.

3.5. Exercises

3.5.1. Multiple choice questions

1. The SJF algorithm is a

i) preemptive CPU scheduling algorithm.
ii) disk scheduling algorithm.
iii) modification of the RR CPU scheduling algorithm.
iv) non-preemptive CPU scheduling algorithm.

3.5.2. Questions for short answers

a) What are the performance criteria of a scheduling algorithm?
b) Write short note on : Throughput, Wait time, Turnaround time, Response time.
c) What are the important factors of scheduling mechanism?
d) What do you understood by preemptive and non-preemptive scheduling.
e) What are the characteristics of FCFS and SJF algorithms? How can they be implemented?

3.5.3. Analytical questions

1) Given the following information

Job	Arrival Time	Service Time
1	0	16
2	1	2
3	2	3
4	3	1
5	4	5

a) Using FCFS

i) Give a Gantt chart illustrating the execution of these jobs.
ii) What is the Turnaround time of job 3 ?
iii) What is the average turnaround time and wait time for all the jobs.

b) Using SJF

i) Give a Gantt chart illustrating the execution of these jobs.
ii) What is the Turnaround time of job 3?
iii) What is the average turnaround time and wait time for all the jobs

c) Complete the following table for each of the following scheduling algorithm

Job	Finish time
1	
2	

3 4 5	

i) FCFS
ii) SJF.

3. Five jobs are in the ready queue waiting to be proceed. The amount of time on the CPU that they require before finishing execution is as follows : 10, 3, 5, 6, 2.

In what order would these processes be scheduled using SJF.

4. Determine the sequence of execution for each of the following CPU scheduling algorithms.

i) Shortest job first
ii) FCFS

Job	Arrival Time	Execution Time
1	0	14
2	3	12
3	5	7
4	7	4
5	19	7

Lesson 4 : Priority, Preemptive and Round Robin Scheduling Algorithms

4.1. Learning Objectives

On completion of this lesson you will know :

- priority scheduling
- indefinite blocking problem and its solution
- preemptive algorithm - shortest Remaining-Time-First
- Round-Robin scheduling algorithm.

4.2. Priority

Shortest-Job-First is a special case of the general priority scheduling algorithm. Priority means that certain elements are given preference over others based on some ranking scheme. A priority is associated with each job, and the CPU is allocated to the job with the highest priority. Equal priority jobs are scheduled FCFS.

The turn around time will be determined by the number of jobs with higher priority that exist in the ready list. Consider the following list of jobs

Job	Service time	Priority
1	40	4
2	20	2
3	50	1
4	30	3

Priority scheduling algorithm produce the following Gantt chart.

Job 3	Job 2	Job 4	Job 1
0	50	70	100	140

The average turnaround time is = $\frac{50+70+100+140}{4} = 90$

The average waiting time = $\frac{0+50+70+100}{4} = 55$

A major problem with priority scheduling algorithms is indefinite blocking or starvation. A process which is ready to run but lacking the CPU can be considered blocked, waiting for the CPU. A priority scheduling algorithm can leave some low-priority processes, waiting indefinitely for the CPU.

Starvation results from the ranking scheme having a problem that means one or more elements are continually overlooked and other elements are given preference.

They are indefinitely postponed or suffer from starvation when they are never the highest ranking element.

A solution to the problem of indefinite of low-priority jobs is aging. Aging is a technique of gradually increasing the priority of jobs that wait in the system for a long time.

4.3. Preemptive Algorithms

Preemptive Algorithms

FCFS, Shortest-Job-First, and priority and priority algorithms, are non-preemptive scheduling algorithms. A process, it can keep the CPU until it wants to release it, either by terminating or by requesting I/O. FCFS is intrinsically non-preemptive, but the other two can be modified to be preemptive algorithms.

Shortest-Job-First may be either preemptive or non-preemptive. A preemptive shortest-Job-First algorithm will preempt the currently executing job, while a non-preemptive Shortest-Job-First algorithm will allow the currently running job to finish its CPU burst. Preemptive Shortest-Job-First is sometimes called Shortest-Remaining-Time-First (SRTF). Preemptive is the counterpart of SJF.

Implementation

Process with the smallest estimated run-time to completion is run next. A running process may be preempted by a new process with a shorter estimate run-time.

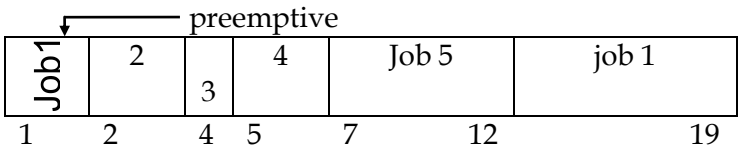
Characteristics

- Preemptive
- Still requires estimates of the future
- Higher overhead than SJF
- Elapsed service times must be recorded
- Possible starvation.

Example
As an example, consider the following 5 jobs.

Job	Arrival Time	Burst Time/Service Time
1	1	8
2	2	2
3	3	1
4	4	2
5	5	5

If the jobs at the ready queue at the times shown and need the indicated times, then the following Gantt chart illustrates the resulting preemptive Shortest-Job-First-schedule.



$$[(19-1) + (4-2) + (5-3) + (7-4) + (12-5)]/5 = \frac{32}{5} = 6.4$$

Job 1 is started at time 1, since it is the only in the queue. Job 2 arrives at time 2. The remaining time for job 1 (7 time units) is larger then the time required by job 2 (2 time units), so job 1 is preempted, and job 2 is scheduled. The average turnaround time for this example is [(9-1) + (12-2) + (10-3) + (14-4) + (19-5)]/5= 9.8 time units. A non-preemptive Shortest-Job-First scheduling would result in an average turnaround time of 9.8.

4.4. Round-Robin

One of the oldest, simplest, fairest and most wide used algorithms is the round-robin (RR) scheduling algorithm, designed especially for time-sharing systems.

One of the oldest, simplest, fairest and most wide used algorithms is the round-robin (RR) scheduling algorithm, designed especially for time-sharing systems.

Characteristics

- preemptive
- effective in time sharing environment
- Penalize I/O bound processes.

Quantum Size

A small unit of time, called a time quantum or time slice, is defined. A time quantum is generally from 10 to 100 milliseconds.

Deciding on the size of quantum is important as it affects the performance of the Round Robin algorithm.

- large or small quantum
- fixed or variable quantum
- same for everyone or different.

If quantum is to large RR degenerates into RCFS.
If quantum is to small context switching becomes the primary job being executed.

A good guide is
"quantum should be slightly larger than the time required for a typical interaction".

Implementation

The ready queue is used to contain all those processes that are ready to be placed on the CPU. 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 up to a quantum in length.

To implement round-robin scheduling, the ready queue is kept as a FIFO queue of processes. New processes are added to the tail of the ready queue. The CPU scheduler picks the first job from the ready queue, sets a timer to interrupt after one time quantum, and dispatches the process.

The process that is currently executing on the CPU will continue until either :

- its quantum expires

Each process has a fixed time limit (the quantum) during which they have the CPU and puts it on the operating system takes the process off the CPU and puts it on the end of the ready queue.

- it is blocked on some event.

The process is awaiting for some event to occur not do any instruction execution until it is finished. The operating system will place the process onto the correct blocked queue. When the waited upon event occurs the process may be put onto either the CPU or the ready queue depending on the specific algorithm used.

Example,
Consider the four processes shown in following table. The time quantum was 20 with a negligible time for context switching. Then, the Gantt chart describing the resulting schedule is shown in Fig. 3.6. The average turnaround time for a process is

$$\frac{40+100+130+140}{4} = \frac{410}{4} = 102.5$$

and the average wait time is

$$\frac{60+20+90+100}{4} = \frac{270}{4} = 67.5$$

0	20	40	60	80	100	120	140
1	2	3	4	1	3	4	3

Fig. 3.6 : Round robin schedule for define q = 20.

Job 1 gets the first 20 time units. Since it requires another 20, it is preempted after the first time quantum, and the CPU is given to the next job in the queue, job 2. Since job 2 needs 20 units, it quits after its time quantum expires. The CPU is then given to the next process, job 3, but it requires another 30, so it is preempted and

CPU is given to the next job in the queue, job 4, but it requires another 10 units, it is preempted, a CPU is returned to job 1 for next time quantum. Then CPU is given to the next job 3, 4 and again job 3 respectively.

Advantages of RR:

- Easy to implement

Disadvantage

- Penalize I/O bound process

Context Switching

We have problems, specifically, at the end of each time quantum, we get an interrupt from the timer. Processing the interrupt to switch the CPU to another process requires saving all the resisters for the old process and loading the registers for the new process. This task is known as a context switch. Context switch time is pure overhead.

So, a context switch is the act of saving the current context on the CPU to memory and then changing the CPU context. The context of the CPU is the collection of CPU registers used to track the status of the current execution. Context switches occur whenever there is an interrupt.

To illustrate the effect of context switch time on the performance of round-robin scheduling, let's assume that we have only one job of 8 time units. If the quantum is 10 time units, the job finishes in less than one time quantum, with no overhead. If the quantum is 5 time units, however, the job requires two quanta, resulting in a context switch. If the time quantum is 1 time unit, then 7 context switches will occur, slowing the execution of the job accordingly (Fig. 3.7).

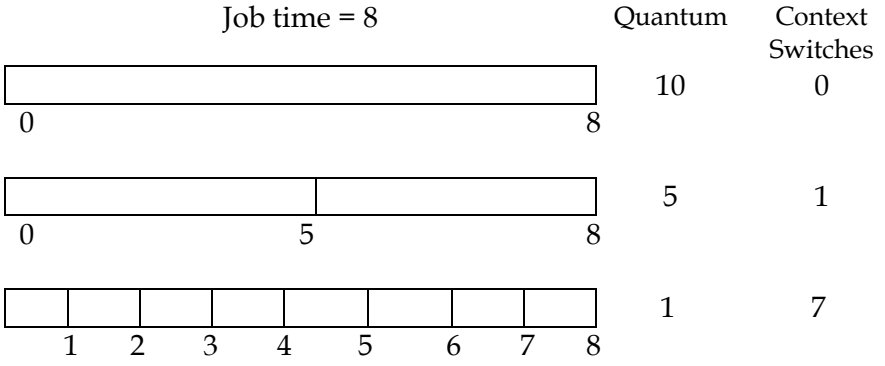


Fig. 3.7 : A smaller time quantum increases context switches.

4.5. Exercise

4.5.1. Multiple choice questions

1. Which of the following in not non-preemptive scheduling algorithm

i) First-Come-First-Served
ii) Priority algorithm
iii) Shortest-Job-First
iv) Round Robin.

2. Round Robin Scheduling algorithm is (select the count algorithm)

i) preemptive
ii) non preemptive
iii) optimal
iv) none of the above.

3. For RR, quantum should be

i) slightly larger than the required time
ii) slightly smaller than required time
iii) equal to required time
iv) none of the above.

4.5.2. Questions for short answers

1. a) What is indefinite blocking? How it is caused?
   b) How may an operating system/algorithm avoid undefined blocking?

2. a) How RR scheduling algorithm works?
   b) What are the advantages and disadvantages of RR scheduling algorithm?
   c) What do you know about Quantum size?

3. a) What is a context switch?
   b) When do they occur?
   c) What is the effect of context switch on RR?

4.5.3. Analytical questions

1. Consider the following list of processes.

Job	Arrival Time	Service Time	Priority
1	1	8	2
2	2	2	4
3	3	1	3
4	4	2	4
5	5	5	1

Answer each of the following questions :

a) using FCFS scheduling.

i) What is the average turnaround time?
ii) What is the turnaround time of Job 4?
iii) What is the average wait time for all Jobs?

b) Using priority algorithm.

i) Give a Gantt chart illustrating the execution of these jobs
ii) What is the average turnaround time and average wait time for each job?
iii) What is the turnaround time for job 3?

c) Using Shortest Remaining-Time-First (SRTF)

i) What is the average turnaround and average wait time for each job?
ii) What is the turnaround time for job 1?

d) Using round robin scheduling with a quantum of 2.

i) What is the average turnaround time and wait time for all the jobs?
ii) What is the turnaround time for process 3?