

Information Security-3
Prof. V Kamakoti
Department of Computer science and Engineering
Indian Institute of Technology Madras
Basics of Unix and Network Administration
Operating Systems Introduction
Mod01 Lecture 07
Module 7: Scheduling Algorithms

So in this module we will talk about scheduling algorithm. So what we completed till the earlier module is that need for a scheduling. So there are three queues as we had seen one is ready queue, one is the running, running is not a queue and there is a waiting queue. So there can be processes, which can be readily scheduled, which are ready for execution and it basically it can be scheduled then there are processes which I have ask for some disc IO, they will be in the waiting queue. The operating system will go and that disc IO or whatever is complete their service is complete then again, it will be moved back to the ready queue and again as scheduling decision will be taken and it will again come back and start executing whenever it is scheduled.

So as a process when I am created, I go to the ready queue then the scheduling algorithm which we have going to see in this module will look at me and if it feels that I should be the next fellow to run, it will make me to the running state, it will allow me to execute on the CPU and then I may ask for say operating system service. Once I ask for that service then I will be moved, there will be a context switch, my process control block will be saved and I will be moved from the running state to a waiting queue, I will wait there till the operating system finishes my service, after the operating system finishes my system service, I will be again put into the ready queue, I will be again put into the global `(())(1:42)` and there again the scheduling algorithm if it finds me suitable, it will again scheduling.

So this is how as a process I will work, but as a process when I am executing, there is an external interrupt, right for some other reason and I am not responsible for it then I am not ask for any service. So immediately I will be moved from the running state to the ready queue and when I moved again my process control block everything will be saved If I am not executing my context will always be in my process control work.

(Refer Slide Time: 2:35)


Criteria For Optimization

12

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

Optimize the average measure, or min/max (i.e., how bad can the "worse-case scenario" be?)

Minimize variance in measure, thus resulting in better predictability — meaningful for interactive systems



Operating Systems

So again I will be in the running queue, again the scheduling algorithm whenever it finds me suitable, it will go and schedule me into the CPU. So this is how the whole thing is working. So the notion of a scheduling algorithm as we had seen in the previous module is that I want to go and maximize throughput, I want to go and maximize several things as we saw. So I want to maximize my CPU utilization, I want to maximize my throughput, I want to minimize my turnaround time, I want to minimize my waiting time, I want to minimize my response time. So for all these criteria the scheduling algorithm will work based on this criterion to select the next process that needs to be scheduled on to this CPU, right.


(Refer Slide Time: 2:52)

Few Things To Remember

16

- One reason for multiprogramming is to improve responsiveness, which means that users will have to wait less (on average) for their jobs to complete.
- The second reason for multiprogramming is to improve hardware utilization.
- We know applications use one system component at a time.
- Multi programming allows simultaneous use of several components
- With Scheduling algorithms both responsiveness and improve hardware utilization issues are addressed.
- Improved utilization leads to improved responsiveness.

All these depends on how you mix the jobs- That is schedule



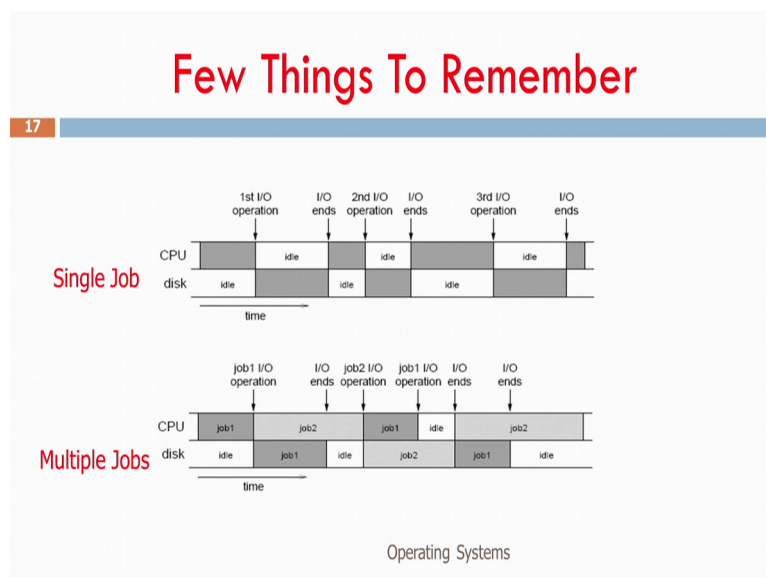
Operating Systems

So some of the few things that we need to remember is that first to one thing one reason for multi programming. Why more than one task need to execute at the same point of time so that all of us all these process wait for less on an average and so I will get more response, right. So it is like multiple fellows login to a mail server, every fellow should get some time on the CPU then only you will feel that you are being served (3:20) by the mail server if one fellow finishes then only the next fellow should start then there is no way by which this responsiveness can be created. So what is multi programing?

The degree of multi-programing is the number of concurrent processes that are admitted into the system. It is a sigma of the number of processes in your ready state and your waiting state plus what is executing. So that is called degree of multi-programing and if the degree of multi programming is large, with a good throughput that means you are going to every program will get a good response from the system.

The second reason for multi programming is to improve hardware utilization as that we have already seen between a disc and CPU we gave some example. Many of the application use only one system component at a time; for example, they use either the CPU or the disc. So if I have multiple components then each one will use one of these and so the entire hardware utilization will go high and the important thing is that I need to have a good mix of jobs we would use different components of the system and I get maximum hardware utilization and also, meet some of the criteria's that we have mentioned in the previous modules.

(Refer Slide Time: 4:32)

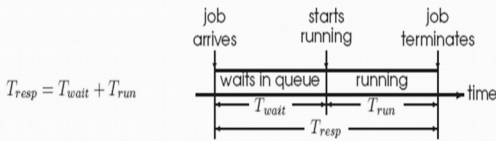


So let us look at the different scheduling algorithms very quickly. Now this is a notion of a single job we have explained this in the previous module, I will just explain it for a sake of you know revision. A single job basically executes on the CPU, it also executes on the disc, sometimes it is on the CPU sometimes on the disc whenever it is on the CPU, the disc is ideal whenever is on the disc, the CPU is ideal, if I have multiple jobs when the first job uses the CPU, the second job can essentially use the disc and when the first job uses the second job uses the CPU, the first job can use the disc. By this please note that the empty white space white space is idle run time then amount of white space in the top in the context of single job what do you see on the screen is much more than the amount of white space that you see in the bottom in the context of multiple jobs, essentially employing that I have better hardware utilization.


(Refer Slide Time: 5:28)

Few Things To Remember

18



Response time :- The average time from submitting a job until it terminates.
This is the sum of the time spent waiting in the queue, and the time actually running:



Operating Systems

Now let us look at some of the timing. So a job arrives and it actually waits in the running queue, it waits in the ready queue and it starts running and it terminates so that amount of response Tresponse time is the average time from submitting a job until it terminates. This is the sum of the times spend waiting in the queue and the time actually running. In the context of real time systems this gets a small deviation (5:55), the response time is the time between time of starting minus the time of arrival, okay. So that is also called the first response time. Generally, people are interested in reducing this response time, because they want the result to be done within as early as possible.

(Refer Slide Time: 6:13)

What does the Scheduler do?

19

- When CPU becomes idle the OS must select one of the processes in the Ready Queue to be executed
- **CPU Scheduler** selects a process from the processes in memory that are ready to execute (from Ready Q), and allocates the CPU to one of them
- The OS code that implements the CPU scheduling algorithm is known as **CPU scheduler**
- **CPU scheduling decisions** may take place when a process:
 - ▣ Switches from running to waiting state
 - ▣ Terminates
 - ▣ Switches from running to ready state. (e.g. when time slice of a process expires or an interrupt occurs)
 - ▣ Switches from waiting to ready state. (e.g. on completion of I/O)
 - ▣ On arrival of a new process

Operating Systems

So the role of a scheduler we had seen the role of the scheduler in great detail in the previous lecture, but I am just (6:20) up is that there is a ready queue and from the ready queue, it has to look at all the processes and it has to select the next process that needs to go and get scheduled on to the CPU and that is the role of the scheduler.

(Refer Slide Time: 6:34)

Scheduling Algorithms

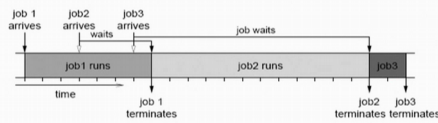
20

- **First-come, first-served scheduling (FCFS) algorithm**
- **Shortest Job First Scheduling (SJF) algorithm**
- **Shortest Remaining time (SRT) algorithm**
- **Non-preemptive priority Scheduling algorithm**
- **Preemptive priority Scheduling algorithm**
- **Round-Robin Scheduling algorithm**
- **Highest Response Ratio Next (HRRN) algorithm**
- **Multilevel Feedback Queue Scheduling algorithm**
- **Multilevel Queue Scheduling algorithm**

Operating Systems

The Case For Scheduling : FCFS

21



- Job 2 is ahead of job 3 in the queue, so when job 1 terminates, job 2 runs. However, job 2 is very long, so job 3 must wait a long time in the queue, even though it itself is short.
- Reason for multiprogramming is to improve responsiveness, which means that users will have to wait less (on average) for their jobs to complete.
- Jobs are serviced in the order they arrive in (First Come First Serve, or FCFS) scheme
 - Short jobs may be stuck behind long ones.

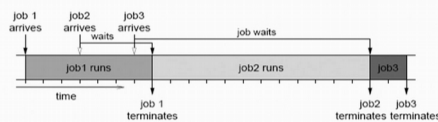
Operating Systems

So there are many scheduling algorithms the first come first serve, the shortest job first, the shortest remaining time first, non-preemptive priority scheduling, preemptive priority scheduling, round robin highest response ratio next, multilevel feedback queue scheduling, multilevel queue scheduling algorithm. So these are some of the different algorithms that we had scheduling algorithms that are of used and we will see some of them as we go.

(Refer Slide Time: 7:05)

The Case For Scheduling : FCFS

21



- Job 2 is ahead of job 3 in the queue, so when job 1 terminates, job 2 runs. However, job 2 is very long, so job 3 must wait a long time in the queue, even though it itself is short.
- Reason for multiprogramming is to improve responsiveness, which means that users will have to wait less (on average) for their jobs to complete.
- Jobs are serviced in the order they arrive in (First Come First Serve, or FCFS) scheme
 - Short jobs may be stuck behind long ones.

Operating Systems

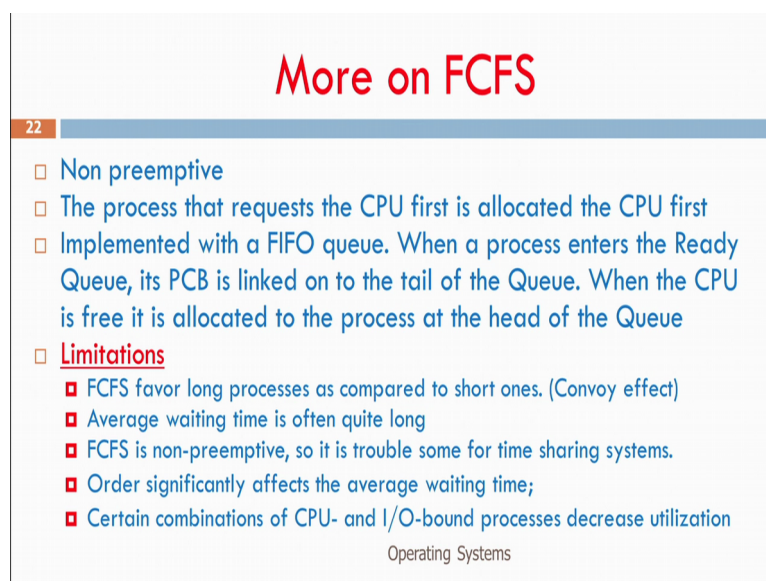
The first come first serve scheduling is a simplest where a job comes, it starts executing please, note see the timing diagram on the top, the job one arise and it is executing by that before it finishes job 2 comes and it waits, before even job 1 finishes job 3 comes and it also starts waiting. Once, job 1 finishes job 2 runs after job 2 finishes job 3 runs. So the overall waiting time is job 1 does not wait for any time here, that job 2 waits for actually you see 4

units of time while job 3 waits for 4 plus 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 16 units of time or 13 units of time.

So the overall waiting time for the job3 is quite high. It waits for 13 units of time to execute two units of time to finish. This is about the first come first serve. So if I have a large process then what happens here is your (8:03) some short processes that are following it, we will have to wait for long. So that is the base basic drawback of first come first serve if every process is of short duration then this responsiveness would be quite high but .

Now you see that job 3 is a very quick job, but may be of very all small jobs are very highly prioritized, but it needs to wait for lot of time at least 3 times its execution time we have (8:32) 4 times is execution time, actually 6 times is execution time, it needs to wait for it to start running from that point job 3 arise from the point it starts executing at least 12 units of time are spent.

(Refer Slide Time: 8:46)



More on FCFS

22

- Non preemptive
- The process that requests the CPU first is allocated the CPU first
- Implemented with a FIFO queue. When a process enters the Ready Queue, its PCB is linked on to the tail of the Queue. When the CPU is free it is allocated to the process at the head of the Queue
- **Limitations**
 - FCFS favor long processes as compared to short ones. (Convoy effect)
 - Average waiting time is often quite long
 - FCFS is non-preemptive, so it is trouble some for time sharing systems.
 - Order significantly affects the average waiting time;
 - Certain combinations of CPU- and I/O-bound processes decrease utilization

Operating Systems

They have (8:46) see first as we see is non-preemptive. So when job 1 gets in it completes then only job 2 comes. So they have seen first is a very good for what you called as batch processing where we submit the job and tomorrow morning we come and take it or but it is not good for time shear in (9:03) the systems where your response time need to be fast.

(Refer Slide Time: 9:06)

Ideal Case

23

Processor is shared among jobs.

- When there are k jobs in the system, they all run simultaneously, but at a rate of $1/k$.
- Short jobs (job 3) does not have to wait for job 2.
- The time it takes is proportional to its own length, increased according to the current load.
- It is impossible to implement this ideal.
 - But this can be approximated by using time slicing.

Operating Systems

Now what is an ideal case of the scheduler rule? So when so look at this when job1 comes it starts executing job 2 comes, it also starts executing, job 3 comes it also start executing. So if job 3 needs say, 10 units of time we should finish of in 10 units. So when there are k jobs in the system they all run simultaneously, but at a rate of one by k . This is what we need to finally simulate, right.

So when job 1 is executing, it is completely owned by the system and job 2 arise then the resources are split between job 1 and job 2. When job 3 arrives then the resources are split between job 1, 2 and 3. When job 3 finishes again the resource has split between job 1 and job 2 and when job 1 finishes the entire resource is given to job 2. So this is how as you see on the top this is an ideal case and these type of scheduling algorithms are basically called as round robin which basically says it gives a time slice. So if I have k processes then it the divides the times slice among this k . it gives one k one by k for each and basically does. So this is basically possible using a concept call round robin scheduling.

(Refer Slide Time: 10:18)

The Metrics?

24

- **Wait time** :- reducing the time a job waits until it runs also reduces its response time. (T_{wait}).
 - ▣ As the system has direct control over the waiting time, but little control over the actual run time, it should focus on the wait time
- **Response ratio or slowdown** :- the ratio of the response time to the actual run time: **slowdown** = T_{resp}/T_{run}
 - ▣ This normalizes all jobs to the same scale: long jobs can wait more, and don't count more than short ones.
- **Throughput** the number of jobs completed in a unit of time.
 - ▣ If there are more jobs completed, there should be more happy users.
- **Utilization** :- the average percentage of the hardware (or the CPU) that is actually used.
 - ▣ If the utilization is high, you are getting more

Operating Systems

So these are all the metrics that we need to see we have a wait time, we have response ratio or slowdown, we have throughput and we have utilization, we have seen the difference between the throughput and utilization in the previous lecture. Throughput is the number of jobs that we finish at a unit time, but utilization is amount of time I use a CPU. Throughput and utilization together will talk more about what we called as the wait time.

The wait time is the amount of time the job waits in the ready queue for it to get scheduled and we need to minimize this wait time and there is also something called response ratio or slowdown which is nothing but the ratio between $T_{response}$ divided by T_{run} time, right. This is the ratio of the response time to the actual run time and this is very very important and this normalizes all jobs to the same scale, right and so by this we could basically get and understanding of how each job should be treated.

So for example if I have long job I can wait a bit more and because and if I am having a short job my response ratio will be a little higher and so I need to finish faster. So this response ratio can be used as a important parameter for me to basically go and make a scheduling decision. So these are all the metrics here.

(Refer Slide Time: 11:42)

Shortest-Job-First (SJF) Scheduling

25

- Requires knowledge of CPU burst durations: give the CPU to the process with the shortest next CPU burst;
- Provably minimizes the average waiting time
- How do we know the length of the next CPU burst?
- Better fit for batch systems (long-term scheduling), where users can assign time limits to jobs
- CPU scheduling can approximate SJF by predicting the next burst length — generally via exponential average
- In pre-emptive flavor, we change processes if the next CPU burst < what's left of the current one

Operating Systems

Now some of the good algorithms that we see is one is shortest job first schedule it ((11:46)) a non-preemptive scheduling, we could have a preemptive flavor of this also. So biggest challenge here is that I need to know so what ((11:54)) so if I have a collection of job, I have to find out which job will finish fastest or which job needs the least amount of time and that shortest job is the one I will go and assign it to the CPU.

Now the biggest challenge here is that I need to know the exact time, the next CPU burst of that program, right. How will I estimate the CPU burst? So there are some stastical methods, were estimating this CPU burst, which the ((12:22)) there are some averaging methods exponential averaging methods which would be doing this and so the shortest job first scheduling is basically a non-preemptive algorithm, which uses stastical methods to basically find out the value of the next CPU burst and the out of all the CPU burst computed for all the processes in the ready queue, I will find the one with the minimum value and I will schedule it next, I could have a preemptive flavor for this shortest job first scheduling algorithm also, where we change the processes if the next CPU burst is less than ((12:58)) left of the current one. So this is very interesting scheduling algorithm.


(Refer Slide Time: 13:07)

SJF and SRTF

26

- When the CPU is available it is assigned to the process that has the smallest next CPU burst
- If two processes have the same length of next CPU bursts, FCFS scheduling is used to break the tie
- Shortest Job First (SJF)
 - It's a non preemptive algorithm.
- Shortest Remaining Time First (SRTF)
 - It's a Preemptive algorithm.

Operating Systems



And there is also has algorithm which is called shortest remaining time first. So these are very nice case study what you see on the left hand side. Now a long job is running, in the shortest job first synthesis (13:14) this non-preemptive and say that start position a long job is actually running and a short job also comes at that point of time, it needs to wait for the long job and then there is next short job as to take. When the long job started executing the short job was not there. So long job got scheduled, because it was the shortest job first and when the short job came in then it had to wait it. It came at the position of start and it had to wait for long and then it starts executing.

So the average wait time as you see is somewhere very high from start, right because the short job actually waited for quite long amount of time right, but in a shortest remaining time first, it is a preemptive algorithm when the short job came at the start position, see on your left hand side the figure below, the short job came at when the long job was executing the short job came and (14:10) once the short job came, immediately it was admitted by the scheduler and the scheduler now went again (14:17) and made another decision of whom it should schedule, at that point it found out that a long job will take more time than the short job. So the remaining time for the long job is much larger than the remaining time for the short job.

So the short job got scheduled first and then the long job. So the average waiting time now got reduced and as you see that the average met in time has reduced in this case. So the shortest remaining time first is a preemptive algorithm while the shortest job first a non-

preemptive algorithm. So this basically gives you the difference between preemption and non-preemption and certainly preemptive algorithms can reduce your average wait time.

So with this we come to the end of scheduling algorithms from a security point of view scheduling algorithms you need to understand what is scheduling and what are the different things that happen during a context switch and that is more important from a security perspective may be there is also some notions of how do you handle certain denial of service things, carefully looking at the scheduler, because the scheduler is one which is going to basically get certain things running up.

So we will discuss about that later in some later course not even in this course, but as of now understanding scheduling is very very important for understanding the operating system (()) (15:38) and also networking and that why we have covered a brief of scheduling algorithms. In the next module, we will deal about something much more important than this called process synchronization in great detail. Thank you.