

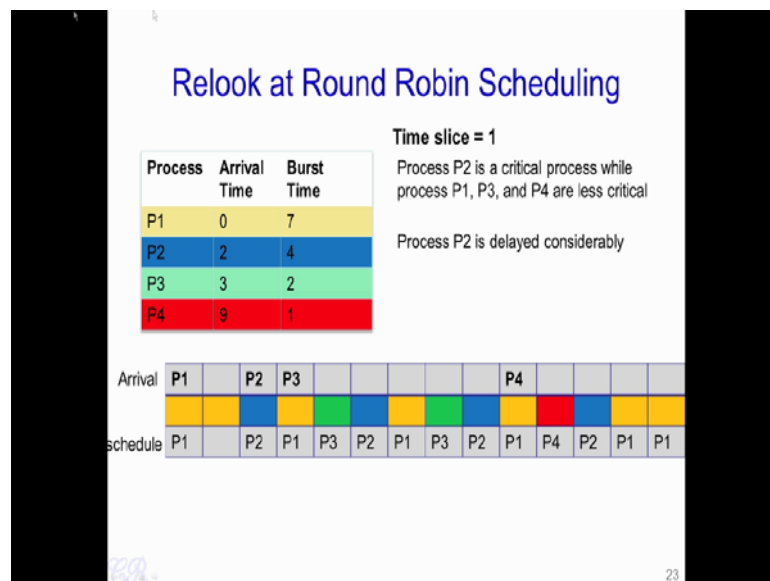
**Introduction to Operating Systems**  
**Prof. Chester Rebeiro**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Week – 05**  
**Lecture – 19**  
**Priority Based Scheduling Algorithms**

So far we had seen about some Scheduling Algorithms like, first come first serve and the preemptive scheduling algorithms like Round Robin.

In this particular video, we will look at a class of algorithms for scheduling which are known as the Priority Based Scheduling Algorithms. We will start this lecture with a motivating example.

(Refer Slide Time: 00:40)



So, let us relook at this particular example which we taken you in the last video. We had seen this Round Robin Scheduling Algorithm where we took four processes P1 to P4, and this processes arrived at different times 0, 2, 3 and 9, and that have different burst times like 7 cycles, 4, 2, and 1 cycles respectively.

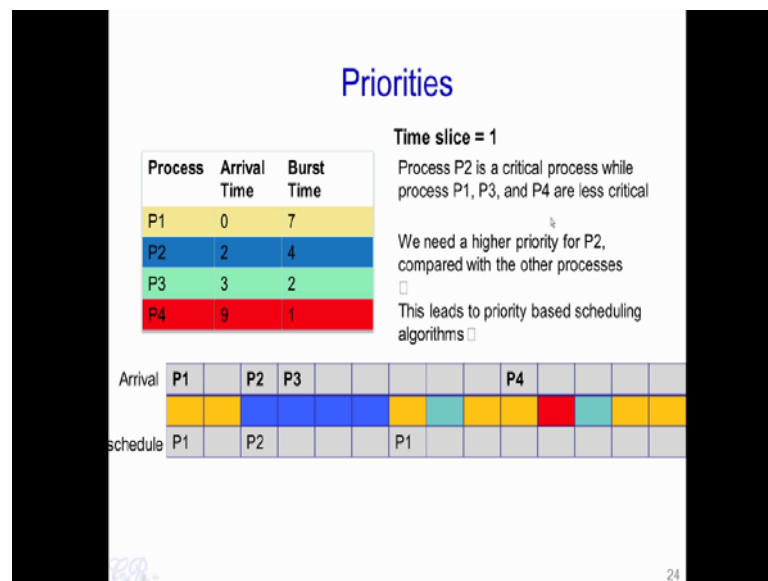
Now we will add a bit above the processes which are involved. Let us assume that P2 is a critical process, while P1 P3 and P4 are less critical. So, what is a critical process? Let us take an example of an operating system that runs in micro processor present in a car, and

whenever the breaks are pressed process P2 will execute and it will do several tasks related to breaking the of the car, while process P1 P3 and P4 may be less critical tasks. With respect to the car again for instance process P1 P3 and P4 may be the music player, so one of the processes or may be the music player while another one could be for instance controlling the AC or the heater and so on.

So, lets us look at this Gantt chart again, we see that P2 arrives in the second cycle and it takes considerable amount of cycles to complete executing. The main reason for this is that this is the round robin scheduling scheme and other processes have a fair share of the CPU. So, P2 first executes here and then there is a P1 and P3 executes and then P2 executes again and again and finally completes executing its burst at its particular time.

Now, you will see that there is an obvious problem with this particular scheme. Since, P2 is a critical process such as controlling the breaking of the car it has taken really long time for it to complete its execution, and this could lead to cryptographic accidents.

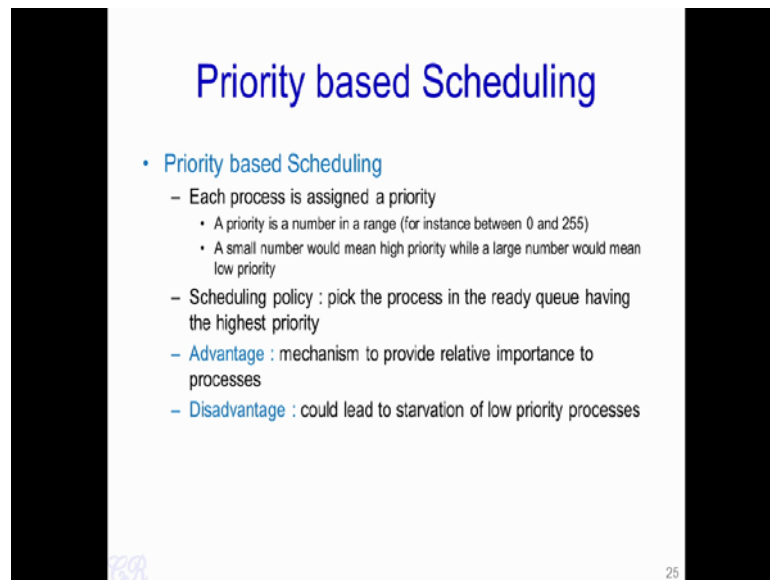
(Refer Slide Time: 02:49)



So what one would have expected is something like this. Whenever a critical process such as P2 arrives in the ready queue, it should get a priority and it should be able to execute continuously, irrespective of the fact that are other processes with the lower priority present in the queue.

This is what we would like to have in a more realistic situation. Scheduling algorithms which take care of such priorities among processes are known as Priority Based Scheduling Algorithms, and this is what we are going to study in this particular video lecture.

(Refer Slide Time: 03:28)



**Priority based Scheduling**

- Priority based Scheduling
  - Each process is assigned a priority
    - A priority is a number in a range (for instance between 0 and 255)
    - A small number would mean high priority while a large number would mean low priority
  - Scheduling policy : pick the process in the ready queue having the highest priority
  - Advantage : mechanism to provide relative importance to processes
  - Disadvantage : could lead to starvation of low priority processes

25

In a priority based scheduling algorithm each process is assigned a priority number. A priority number is nothing but a number within a particular range, so this range is predefined by the operating system. For instance, the priority range is between 0 and 255. Therefore, every process that is executed would be assigned a priority number between 0 and 255. A number which is small, that is a priority number which is small that is close to 0 would mean that the process is a high priority process or a high priority task. On the other hand, a large priority number would mean that the process has a low priority.

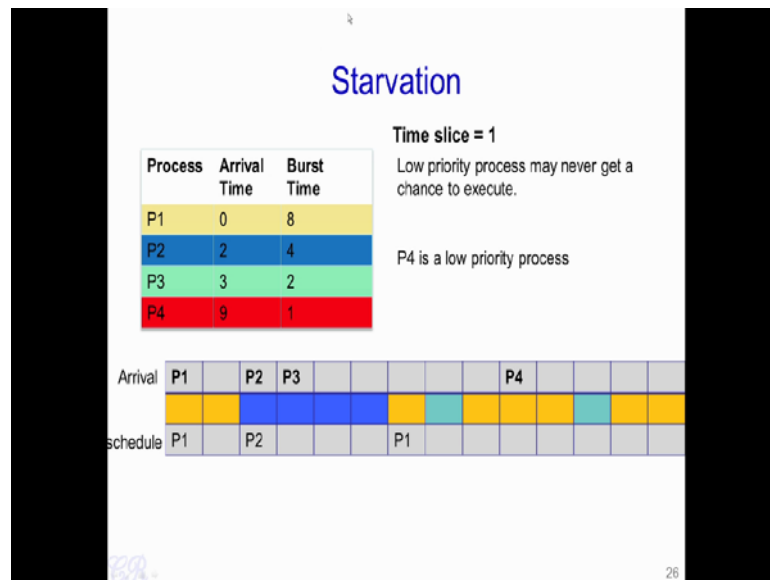
So this just nomenclature which Linux follows, other operating systems may have a different approach. It would keep for instance or higher number indicating a high priority process while a lower priority number indicating a low priority process, but for this lecture we will go with the Linux nomenclature where a small priority number means a high priority process and vice versa.

Now, in a priority based scheduling algorithm even though there may be several tasks present in the ready queue the scheduling algorithm will pick the task with the highest

priority. The advantage of the priority based scheduling algorithms is quite easy to see. Essentially in an operating system which supports a priority based scheduling algorithm, task can be given relative importance or process could be given relative importance. High priority processes would get higher priority to actually execute in the CPU compare to low priority processes.

On the other hand, the main drawback of having a priority based scheduling algorithm is that it could lead to what is known as starvation. So essentially, a star process is one which is not able to execute at all in the CPU, although it is in the ready queue for a long time. It is quite easy to see that starvation mainly occurs for low priority process.

(Refer Slide Time: 05:39)



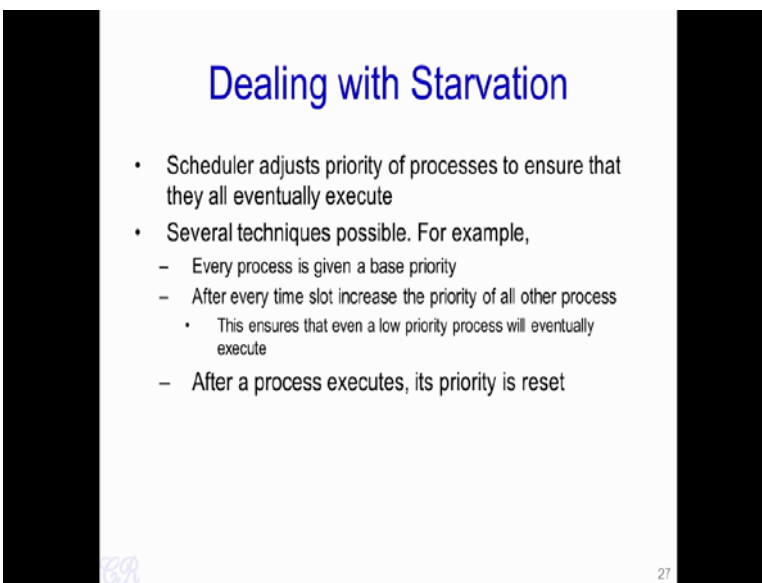
Let us see how starvation occurs with an example. We will take the same four processes P1 to P4 arriving at different times 0, 2, 3 and 9 and having different burst times 8, 4, 2 and 1. Also, let us assume that the priority of P1 P2 and P3 is higher than P4 or in other words P4 is the lowest priority process. We will also make this assumption that every 15 cycles this particular sequence repeats, that is after the 15th cycle of the processor P1 again arrives P2 at again arrives P3 and similarly P4 arrives, this sequence of processes arriving into the ready state continuous infinitely.

So what we see, if you look at the Gantt chart is that depending on the priority P1 P2 and P3 are scheduled. However, the low priority task never gets scheduled within the 15 cycles. Additionally, since P1 P2 P3 and P4 arrive again into the ready queue after 15

cycles. So, again P1 P2 and P3 get scheduled while P4 never get scheduled. As a result what we see is that, this lowest priority process gets starved.

Essentially, even though it is in the ready queue and just waiting for the CPU to be allocated to it, it never gets a chance to execute in the CPU because there are always higher priority processes which are arriving in the ready queue.

(Refer Slide Time: 07:28)



The slide is titled "Dealing with Starvation" in blue text. It contains a bulleted list of techniques. The first bullet point states that the scheduler adjusts the priority of processes to ensure they all eventually execute. The second bullet point lists several techniques: every process is given a base priority; after every time slot, the priority of all other processes is increased, which ensures that even a low priority process will eventually execute; and after a process executes, its priority is reset. The slide is flanked by two vertical black bars. A small number "27" is visible in the bottom right corner of the slide area.

### Dealing with Starvation

- Scheduler adjusts priority of processes to ensure that they all eventually execute
- Several techniques possible. For example,
  - Every process is given a base priority
  - After every time slot increase the priority of all other process
    - This ensures that even a low priority process will eventually execute
  - After a process executes, its priority is reset

Let us see how priority based scheduling algorithms deal with starvation. Essentially, in operating systems which have a priority based scheduling algorithm, the scheduler would dynamically adjust the priority of the process to ensure that all processes would eventually execute in the CPU. What this mean is that, in our example the priority of process P4 would gradually increase over a period of time until the point that it has a priority which is greater than or equal to the priority of the processes P1 P2 and P3, at that time process P4 will get a chance to execute in the CPU.

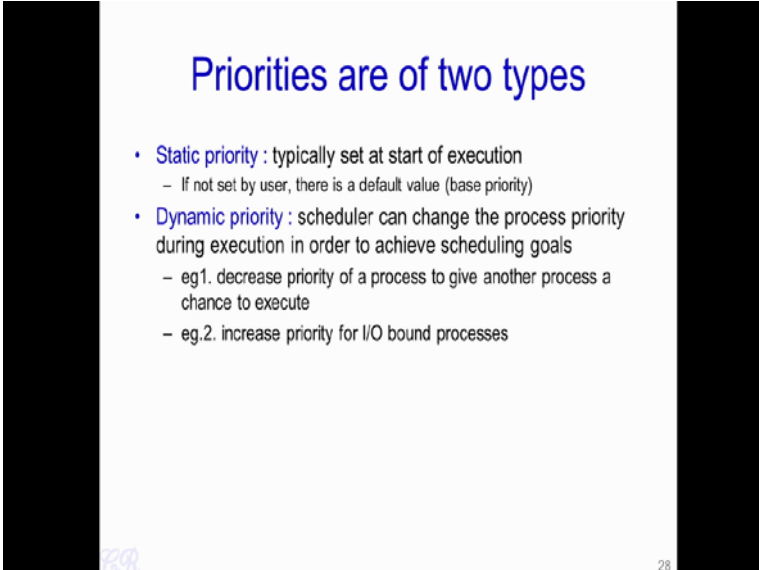
Thus, we see by elevating process P4 over time the starvation that would otherwise have occurred, since P4 was a low priority process is eliminated. There are several techniques that priority based scheduling algorithm could deal with starvation. We give one particular example here in which an operating system with a priority based schedule algorithm could ensure that every process even the low priority process will eventually execute in the CPU. So, let us say in this particular operating system when a process starts it is given a based priority. Now, this based priority will defer from process to

process, a high priority process will be given high priority number, while a low priority process will be given a low priority number.

Now, after every time slot or time slice the priority of the process is increased by a fixed value. So, essentially all processes in the ready queue have their priority increased by a fixed value except for the process that is currently being executed in the CPU. What you would see that, over a period of time all process in the priority queue would gradually have the priority increased. Now even a low priority process would have its priority increased up to the point that its priority is high enough so that it is scheduled the CPU.

So, after it is scheduled into the CPU and execute in the CPU at the end of its time slice its priority is reset back to its based priority. Thus the process which starts up with the base priority of 250 will gradually have its priority increased, so it is be to a point that is begins to execute and after its executes its priority is reset to 250. Thus, starvation would be avoided.

(Refer Slide Time: 10:20)



The slide is titled "Priorities are of two types" in blue text. It contains two main bullet points: "Static priority" and "Dynamic priority". The "Static priority" bullet point has a sub-bullet: "If not set by user, there is a default value (base priority)". The "Dynamic priority" bullet point has two sub-bullets: "decrease priority of a process to give another process a chance to execute" and "increase priority for I/O bound processes". The slide number "28" is in the bottom right corner.

- **Static priority** : typically set at start of execution
  - If not set by user, there is a default value (base priority)
- **Dynamic priority** : scheduler can change the process priority during execution in order to achieve scheduling goals
  - eg1. decrease priority of a process to give another process a chance to execute
  - eg.2. increase priority for I/O bound processes

So based on this, there are two types of priorities present for a process. There is a static priority which is typically set at the start of execution, so it can be set by user who is starting that application, and by chance if the user does not set that particular priority then a default value is taken.

On the other hand, the second type of priority is known as the Dynamic Priority, where the scheduler can change the process priority during the execution in order to achieve some scheduling goals. For example, the scheduler could decide to decrease the priority of a process in order to give another process a chance to execute. Another example as you seen before is to increase the priority of IO bound processes. Since, IO bound processes would typically require a faster response time.

(Refer Slide Time: 11:19)

The slide is titled "Priority based Scheduling with large number of processes". It contains a bullet point stating: "Several processes get assigned the same base priority" followed by a sub-bullet: "Scheduling begins to behave more like round robin". Below this is a table with four columns: Process, Arrival Time, Burst Time, and Priority. The table lists four processes: P1 (Arrival Time 0, Burst Time 8, Priority 1), P2 (Arrival Time 2, Burst Time 4, Priority 1), P3 (Arrival Time 3, Burst Time 2, Priority 1), and P4 (Arrival Time 9, Burst Time 1, Priority 1). The priority for all processes is 1. The slide number 29 is visible in the bottom right corner.

| Process | Arrival Time | Burst Time | Priority |
|---------|--------------|------------|----------|
| P1      | 0            | 8          | 1        |
| P2      | 2            | 4          | 1        |
| P3      | 3            | 2          | 1        |
| P4      | 9            | 1          | 1        |

One disadvantage of having a range of numbers from which a process obtains its priority is the case when there is large number of processes present in the system. In such a case, there may be several process which are executing with the exactly the same priority. For instance, let us say that a priority range is from 0 to 255 and there are may be over all ten thousand processes running in the system, in such a case each priority number there may be several processes executing with that priority number.

So this is depicted in this particular figure, where we have four processes arriving at different times and having different burst times, but all of them have the same priority. In such a case, the scheduling would begin to behave like non priority based scheduling algorithm. For instance it would begin to behave like a round robin scheduling algorithm.

(Refer Slide Time: 12:22)

## Multilevel Queues

- Processes assigned to a priority classes
- Each class has its own ready queue
- Scheduler picks the highest priority queue (class) which has at least one ready process
- Selection of a process within the class could have its own policy
  - Typically round robin (but can be changed)
  - High priority classes can implement first come first serve in order to ensure quick response time for critical tasks

30

In order to handle this is what modern operating systems have is Multilevel Queues that is instead of assigning a single priority number to each process; processes are assigned a priority class. Now there could be several priority classes within the operating system and each of these priority classes could range from a high priority to a low priority. For instance, there could be a real time priority class, a system priority class and interactive priority class or batch priority class and so on.

All real time processes would be present in this priority class. So each of these priority classes has a ready queue and when a real time process is ready to execute it gets added on to this particular ready queue, that is it gets added on to the ready queue present in the real time class priority class.

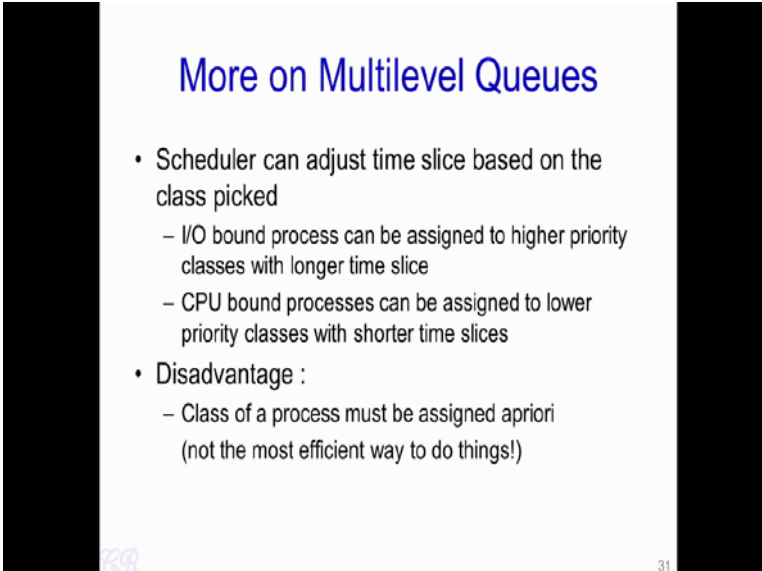
Similarly, if an interactive process wants to execute and it is ready to execute it gets added to this ready queue present in the interactive priority class. Now, at the end of a time slice when the scheduler executes, the scheduler picks the highest priority class which has at least one ready process. For instance, the scheduler will scan through this priority class starting from most high priority to the least priority, and it will select that particular priority class which has at least one process in the ready queue. Now, if there is exactly one process in the ready queue then we do not have a problem, now that process is going to execute in the CPU.



However, if we have multiple processes in the ready queue corresponding to the ready queue chosen by the scheduler, then a second scheduling algorithm would be used to pick or to choose from this set of processes. For instance, let us say the scheduler has decided to choose the interactive preemptive class, because there were no other processes ready in the real time and the system priority classes.

Further, let us assume that there are 4 processes present in the interactive class. Now the second scheduling algorithm such as a Round Robin or a FIFO scheduling algorithm will then execute to determine which of these four processes will be assigned the CPU. Thus, in multilevel queues there are two scheduling algorithm involved. The first is a priority based scheduling algorithm, which chooses one of the priority classes, while the second is non priority based algorithm, which chooses among the processes in that priority class. The second scheduling algorithm would typically the round robin, while for high priority processes the second scheduling algorithm is typically something like first come first serve and so on.

(Refer Slide Time: 15:36)



**More on Multilevel Queues**

- Scheduler can adjust time slice based on the class picked
  - I/O bound process can be assigned to higher priority classes with longer time slice
  - CPU bound processes can be assigned to lower priority classes with shorter time slices
- Disadvantage :
  - Class of a process must be assigned a priori (not the most efficient way to do things!)

31

Further, in schedulers that support multilevel queues, the scheduler could adjust the time slice based on the priority class that is picked. So, IO bound processes are typically assigned a higher priority class and a longer time slice. This ensures that the IO bound processes would be quickly serviced by the CPU as soon as possible. Also the longer

time slice will ensure that the burst of the IO bound process completes before the time slice completes.

CPU bound processes on the other hand are assigned a lower priority class and given shorter time slices. The main drawback of having a multilevel scheduling queue based scheduling algorithm, is that the class of the process must be assigned a priori that is we would require to know whether a process is a CPU bound process or an IO bound process. Now this is not an easy thing to do; first it is very difficult to identify whether a process is indeed or an IO bound process or a CPU bound process. Second, a single process may act like an IO bound process at one time, but at another time it can act like a CPU bound process.

That is, at one time it may have short CPU burst and long burst of idle time, while at other times it may have long CPU burst and short IO burst like a CPU bound process. Therefore, deciding a priori whether a particular process belongs to IO bound process and it should given a higher priority is difficult.

(Refer Slide Time: 17:24)

**Multilevel feedback Queues**

- Process dynamically moves between priority classes based on its CPU/ IO activity
- Basic observation
  - CPU bound process' likely to complete its entire timeslice
  - IO bound process' may not complete the entire time slice

Process 1 and 4 likely CPU bound  
Process 2 likely IO bound

32

So, one way to mitigate this particular limitation is to use schedulers which use multilevel feedback queues. In such schedulers' process are dynamically moved between priority classes depending on its CPU, IO activity. These particular schedulers work on this basic observation that CPU bound processes are likely to compute its entire time

slice. This is because a CPU bound process will have a long burst of CPU and therefore it is quite likely that it would use of entire time slice.

On the other hand IO bound process has a very short CPU burst time and therefore it is not likely to complete its entire time slice. If you take this particular example, we can say that process 1 and 4 have completed their time slice; therefore these processes are likely to be CPU bound. On the other hand, process 2 has very short CPU burst over here as well as here, so therefore process 2 is likely to be an IO bound process rather than a CPU bound process.

(Refer Slide Time: 18:41)

### Multilevel feedback Queues (basic Idea)

- All processes start in the highest priority class
- If it finishes its time slice (likely CPU bound)
  - Move to the next lower priority class
- If it does not finish its time slice (likely IO bound)
  - Keep it on the same priority class
- As with any other priority based scheduling scheme, starvation needs to be dealt with

33

The basic idea of a multilevel feedback queues based scheduling algorithm is the following; all processes start with the highest priority or a based priority. Now if the process finishes its time slice, that is if the process executes until its time slice completes then the assumption is made that process is a CPU bound process. It is moved to the next lower priority in the class.

On the other hand if the process does not completes its time slice, it is assume that the process is an IO bound process and therefore it would either increase the priority class to a higher priority class or keep it on the same priority class. This as you can see is a dynamic way of changing priority classes of a process. However, the drawback is that we could still have starvation and then techniques are to be implemented where starvation needs to dealt with.

(Refer Slide Time: 19:46)

### Gaming the System

- A compute intensive process can trick the scheduler and remain in the high priority queue (class)

```
while(1){
do some work for most of the time slice
sleep(till the end of the time slice)
}
```

Sleep will force a context switch

Process 4 is gaming the system

time

34

Another drawback of this particular scheduling algorithm is that, malicious process could game the system and always remain in the high priority class. So, let us see how this works with an example. Let us assume the malicious process knows the time slice of the system that is it knows whenever the time slice completes and it knows when a context switch occurs. Let us say it has a loop such as list over here. Essentially, the malicious process will do some work for most of the time slice and then it will sleep till the end of the time slice.

Now as we know when a process goes to sleep, it goes from the running state to the block state. Therefore, sleep will force a context switch to occur. Therefore, what is going to happen is that a new process will execute for the remaining of the time slice. Thus, for the entire time slice for instance over here process 4 which is the malicious process would run for most of the time slice and then just before the time slice completes it goes to sleep; thus, forcing and other process to execute.

The other process 1 will now execute and just execute for a small duration. At the end of the time slice the scheduler is going to see that process 1 has completed the time slice, so it is going to assume that process 1 is a CPU bound process and move the process 1 a priority class which is lower.

On the other hand, process 4 which has blocked on a sleep will remain in the high priority class. Thus, process 4 will be able to game the system. If we will continue to

execute from the high priority class while another process such as process 1 will be moved on to a lower priority class.

Thank you.