

Operating System Fundamentals
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 13
Processes (Contd.)

(Refer Slide Time: 00:33)

Process Control Block (PCB)
Information associated with each process
(also called **task control block**)

- Process state – running, waiting, etc
- Program counter – location of instruction to next execute
- CPU registers – contents of all process-centric registers
- CPU scheduling information- priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files

process state
process number → PID
✓ program counter
✓ registers
✓ memory limits
✓ list of open files
...

Stack
Data
Code

PC
PC offset

So, in our last class we were discussing on the Process Control Block or PCB. So, as we said previously that when a process is executing, so it has got a certain snapshots from the memory and those snapshot information is very important because if you have to suspend the execution of the process for some time and we want to restart it later then the same environment has to be given. For example, if I say that I have got a piece of program which is consist say this is a program that we have and this program has got several lines in it.

Now; so maybe at present we are at this line of execution at which are some requirement came from the system. So, that this process has to be suspended and some higher priority process that has started that has come, so that has to start. Then what happens is that after that higher priority process is over I must be able to come back with execution of this particular process and it has to start from this point onwards, so it cannot be from this.

Now, the things that we need to remember at this point. So, that we can restart from this, one is the program counter value for the process and also when a process is executing it

may have a number of variables, and some of these variables are allocated space in the global data segment, and some of them are allocated in the local data stack, and some of them are allocated in to the CPU registers. So as far as the CPU is concerned this program counter plus CPU registers, so these are important.

And if this particular program, so it has been given different parts in the memory may be in this is the code segment for this program. So this is the data segment for the program and this is the stack segment for the program, so it may be like this. So I need to remember the regions of the memory which has got this code data and stack for this particular process. So at this point of time so what are the code data and stack segment limits for this particular program that has to be remembered.

So that later on when the process comes back so we can again set this limits and we can load appropriate CPU registered that needs to know the values of this limits and the program counter and the CPU registers so all those values can be restored.

Now how do you remember all this things and this Program Control Block or PCB comes to help in this particular situation. Because when I am supposed I have executed up to this and then I have to leave the CPU then this program counter CPU registers and his memory limits similar they are remember.

Similarly if the process has opened some files then this list of open files is also remember. So many other information will be required and which information you should put in the PCB. So that is a bit designer specific like some which designer feel that some information should be kept in the PCB where as the others may feel no it is not necessary. But in general number these are the most important thing that we have; one is the state of the process, then the process number. So, this is very commonly known as PID or Process Identification Number.

So, this is a very important and unique number for every process this PID is unique and whenever the process is created for the first time that it is in the state new. So this PID one PID is assigned to the process. So this way this process control block will have this information. It also keep some more information like this CPU scheduling related information, like I may have a policy that if a process is coming from say user 1 it will have a higher priority than a process coming from user 2.

So for the scheduling information I should have this priority information available. I can also have a policy like this that if a system is using lot of CPU's space CPU time, then its priority should be low compared to a process which uses less CPU time. So, this is basically giving a priority to a interactive program then a computation oriented program. So, interactive program means, so it will be taking lot of data from the user or the file etcetera.

So that way; that has to given priority because otherwise the user will feel that the system is not doing my job. So, this interactive job's they definitely have higher priority than this jobs which are computation oriented jobs. So like, how do you know whether a task is compute interactive or competition oriented.

So, that is basically done by monitoring the how much time the CPU is being used by the process. So, if the CPU is; if the process is given CPU for 2 second and we find that the processors used entire two second without going for an I O block. So, that means, the process is called the compute bound job and if it is the other way that is out of these 2 second may be it has executed for point one second and then it has done for an I O block; that means, it is an I oriented jobs.

So, that way this information has to be stored. Now, where to store this information for the process? Naturally, PCB is the answer. So, you can keep the CPU scheduling information in to the PCB, then memory management information. So what are the memory locations located for this particular process? So that is important; so how much because many times we need to protect the memory space.

So, that it is not access price other processes, many a times we need to have some charging corresponding to that memory usage. So, if you process uses more memory so, it will be have; it has the user has to pay more for the information. So that way we can have some memory management related information like memory allocated to the process. Then accounting information.

So, accounting information like the CPU, how much time the CPU is used, what is the actual clock time from the start time, then time limits? So, like that maybe some processes we can start only at some particular time. For example, if there is some badge job may be the badge jobs they will be done at night only. So, during daytime we are running only the interactive job.

So, this type of time limit; time related information, so there should be kept in to the PCB. So, this accounting information is kept, then the I O status information, so I O devices allocated to process then list of open files etcetera. So, they are to be remembered because; so, if you have got a process which has opened a file, then if some other process also wants to modify that file then that should not be allowed. Even if this process is not executing now, so the files that are captured by that are currently opened by this process should not be given to other processes to access. So, you should have a note of the list of open files for the current process.

So, in this is way this process control block, so it holds the majority of the operation; majority of the information that we need for running the process smoothly. So, and if you want to switch from one process to another then this PCB is going to help us in big way.

(Refer Slide Time: 08:07)

Threads

- So far, process has a single thread of execution
- Consider having multiple program counters per process
 - Multiple locations can execute at once
 - Multiple threads of control -> **threads**
- Need storage for thread details, multiple program counters in PCB
- Covered in the next chapter

The slide features several hand-drawn diagrams: a vertical stack of horizontal lines on the right, a box with three horizontal arrows pointing right in the center, and a diagram labeled 'Web server' with three ovals and arrows pointing to them on the right. A presenter is visible in the bottom right corner of the slide.

So, all operating systems they maintain PCB in some form or the other. So, next will be discussing something on threads. So, we will be coming back to this threads in more detail later. So, in this particular slide we just try to understand it to some concepts about thread. So, process has got a single thread of execution. Now, what is happening is that if for example, the roots of a quadratic equation that program. So, if these are the program execution lines. So, it starts from one end and goes to the other end; so, it follows there is only one thread of execution. So, for this type of programs we cannot have different

request, but consider the situation where I have got a web server, and this web server so, different users, so they are sending request to this web server for different pages.

Now, if the web server does it like this that it takes the request on the first one, then it processes it and then it goes to the request for the second one. So, if it does like this than the person who has at the end, so we will feel that my requests are not being served. So, what this web server does in state is that this web server, so it creates a thread that takes care of this particular request. Similarly, it creates another threads that care takes care of this.

So, this has got the advantage because even if this thread is due to some problem or the other this thread is halted or this faces some problem. So, other threads, so they will continue. So, that way we have got this thread based design. So, we have got to multi. So, we have; so within the same program we can have multiple execution sequences.

So, this process has got it in the process till now got a single thread of execution. Now, if I have got multiple program counter spark process. So, each thread is executing the same routine for this web browsing, but this thread maybe executing at this location, where the second thread may be somewhere here, third thread may be somewhere here. So, they are at different program counter values. So, this multiple locations can execute at once and multiple threads of control, so they are called threads.

So, need storage for thread details and multiple program counters in PCB. So, we need to augment the PCB structure where it keeps a note of the threads that are currently there in the system for this particular process, what are the corresponding program counter values and what are the corresponding CPU register value, so they are to be remembered.

So, we will come back to this thread discussion in detail later in our later part of our course, but as per as PCB is concerned for every thread I have some information stored in the PCB of the process.

(Refer Slide Time: 10:58)

The slide is titled "Process Representation in Linux" and is subtitled "Represented by the C structure task_struct". It displays the following C code snippet:

```
pid_t pid; /* process identifier */
long state; /* state of the process */
unsigned int time_slice /* scheduling information */
struct task_struct *parent; /* this process's parent */
struct list_head children; /* this process's children */
struct files_struct *files; /* list of open files */
struct mm_struct *mm; /* address space of this process */
```

Below the code, there is a diagram showing three boxes labeled "struct task_struct process information". Arrows indicate a parent-child relationship between them. One box is labeled "current (currently executing process)". To the right, a hand-drawn tree diagram shows a root node labeled "init 0" with arrows pointing to child nodes, one of which is labeled "parent" and another "children".

Now, if you look into this Linux operating system, then this process representation is by means of a structure called task and this structure task, it has got a number of components in it; one is the pid task, pid which is a unique number. So, it starts with the 0 and goes on increasing to a maximum value and that it comes back to some 0 again.

Then we have got a long variable state, so it is the state of the process. Then we have got the time slice for scheduling information, how much time slice is the process is given CPU, then for how much time it should be given. And there may be different depending upon the priority this time slice value may differ, some process we may want to give less time slice, some of them we want to give high more time slice. So, that information may be coming.

Then we have got the parent of the process, so particularly in Linux and Unix operating system we have got this concept of hierarchy and there is a parent child relationship between the processes. So, we have got the first process which is known as the init. So, this is the first process created and this is process pid is 0 and all other process is they are created from this init process ok.

So, any process at any point of time, so it can create a new process and then that new process can again give rise to a number of new processes. So, you have got these parent child relationships. So, if this is the parent we have called the called this has the

child and for these processes this is the parent and they are the children processes. So, we have got the parent child relationship between the processes.

Now, if this parent dies at some time may be after due to some erroneous situation suppose this parent dies, then what happens to this child which become orphan. So, what this Linux or Unix says is that this becomes now a child of the init process. So, that way we have got a parent child relationship and this we have got for every process we remember the parent and the children list.

So, this is the current process; so this is the currently executing process. So, then we have got this; we have got a task chain actually. So, these are the task chain these are task chain, so we have got their all of them are put in to a chain like that. Then we have got this children they are putting on they are put on a list, then we have got files list of open files and address space for this process this structure mm. So, this way we can have we have got information available in the Linux operating system.

(Refer Slide Time: 14:04)

The slide is titled "Process Scheduling" and contains the following text:

- Maximize CPU use
 - Quickly switch processes onto CPU for time sharing
- Process "gives" up then CPU under two conditions:
 - I/O request
 - After N units of time have elapsed (need a timer)
- Once a process gives up the CPU it is added to the "ready queue"
- **Process scheduler** selects among available processes in the ready queue for next execution on CPU

Handwritten on the slide is a diagram showing a circle labeled "Ready" with a dashed arrow pointing to a circle labeled "Running". A small box with a vertical bar is also present near the transition.

So, other operating system will also have similar such structure. So, if you look into say windows or any other operating systems, so we will find that the similar such process structure is maintained; PCB structure is maintained. Now, process; so this is a very important concept because we have got at any point of time a number of processes which are ready for execution. And this operating system it has to decide like which process

should execute next ok. Now, how to do this? So, what are the objectives behind doing this process scheduling? One is definitely to maximize the CPU usage.

So, will like to have the CPU utilized for 100 percent time so, but this 100 percent time is often not possible because maybe one process was given CPU. So, it executed for some time and then it is asking for some I O operation. So, when it asks for IO operation, so we have said as a general policy we put the process into a block state and we take up another process for execution. But in between what happens is that putting this process out of CPU and taking the next process into the CPU, so that take some time.

So, that time the CPU is not being properly utilized. So, we say that this switching if it is can be made very fast. So, then this time will be less and so we can maximize the CPU time. So, if I one possibilities that we can do a scheduling, so that this number of switching's will be less and other thing is that we can make the switching's faster. So, that the time needed for a particular switching is not much.

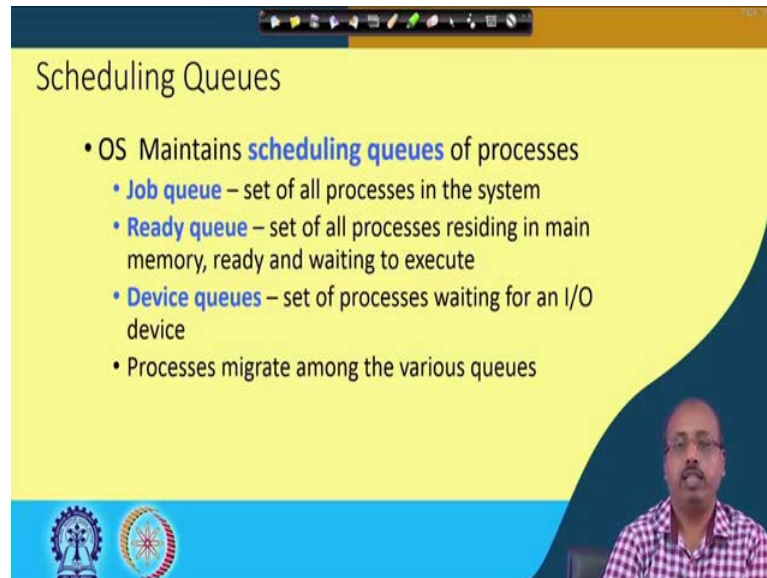
So, we can maximize the CPU is it, so by quickly switch process is onto CPU for time sharing. So, process gives up then CPU under to give some the CPU under two condition one is the I/O request that I have already said that there is some I/O operation and that has to be done and after N units of time have elapsed. So, there may be a timer and then after the process was given some time slice for execution and after that time slice expires the process is taken back from the CPU and so that way the process can give up CPU.

And once the process gives the CPU it is added to the ready queue. So, once it gives up, so it is joining the ready queue and if it is waiting for some IO event so it will join the I O event queue. But any way the process will be put into a queue again, and this process scheduler it has to select among available processes in the ready queue for next execution. So we can ideally see it like this. So if this is that ready state and this is a running state now this CPU with this the scheduler. So we can say that there is a queue associated with this ready so any process which is joining in the ready queue, so they are actually joining this queue.

Now depending upon the priority of all this jobs that we have in the queue and other rescheduling related parameters this schedulers so it will pick up one particular job from this and it will put it into the running state. So that was the dispatch process those the

scheduling decision has to be proper. So that the process is taken from the ready queue and it is put on to the running state.

(Refer Slide Time: 17:25)



The slide is titled "Scheduling Queues" and features a yellow background with a blue wave-like shape on the right side. At the top, there is a navigation bar with various icons. The main content consists of a bulleted list:

- OS 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

In the bottom right corner, there is a small video inset showing a man with glasses and a mustache, wearing a checkered shirt, speaking. In the bottom left corner, there are two circular logos: one with a gear and a person, and another with a sun-like symbol.

So this way there can be different types of scheduling queues. So this queues maintains scheduling queues the, so we have got job queue setup all processes in the system. So, whatever is processes are there in the system so everything is there in the job queue. Then there is a ready queue which is the set of all processes residing in the main memory and ready for execution. So there otherwise ready in the memory so, if they get a chance for execution they get the CPU then they will exit the process will execute, so they are put on to the ready queue.

Then we have got device queues; so device queues so, they are set of processes waiting for an IO device. So for example, may be there are request to the disk and there are many processes which are requesting to the disk. Now disk being a semiconductor, so sorry this is being an electromechanical device. So, it is not as fast as the semiconductor devices.

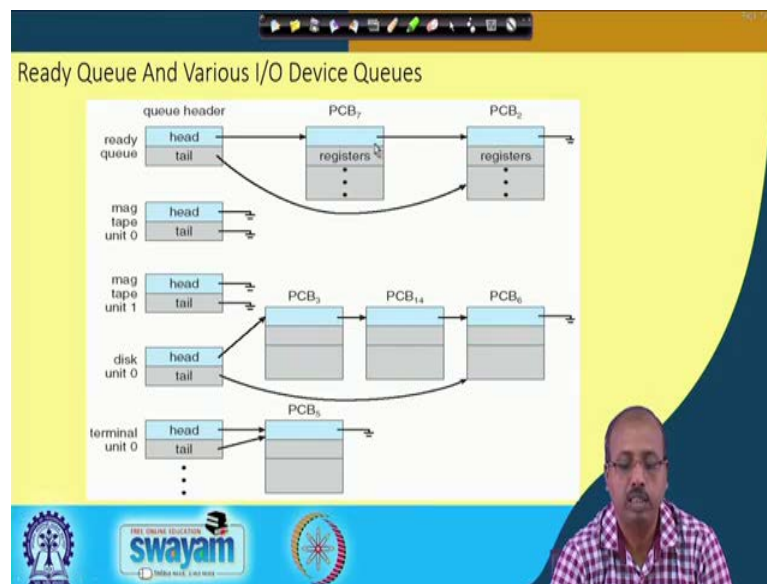
So as a result it will take some time to service those request. So the disk head it needs some time to go from one sector to another sector from one track to another track. So, that way it requires some time. So, other processes which have requested for this IO operation they have to wait ok. So this waiting time may be variable depending upon the

type of resource that we are talking about and the load on that particular device how many users are how many processors are waiting for accessing that particular device.

So this device queues will be there and processes the migrate among various queue. So, initially the process when the process is created, so it is put into the job queue from there when the process has got the CPU got the copied onto main memory, it comes to the ready queue.

And then from the ready queue so depending upon the scheduling so it may go to the running state or after running for sometime it may join the ready queue or it may request for some IO operation and join the corresponding device queue. And from the device queue if the job is taken up for IO satisfaction and the IO operation is done, then it will be joining back to the ready queue and that way it goes on. So, that is why a process can migrate among different queues in the system.

(Refer Slide Time: 19:45)

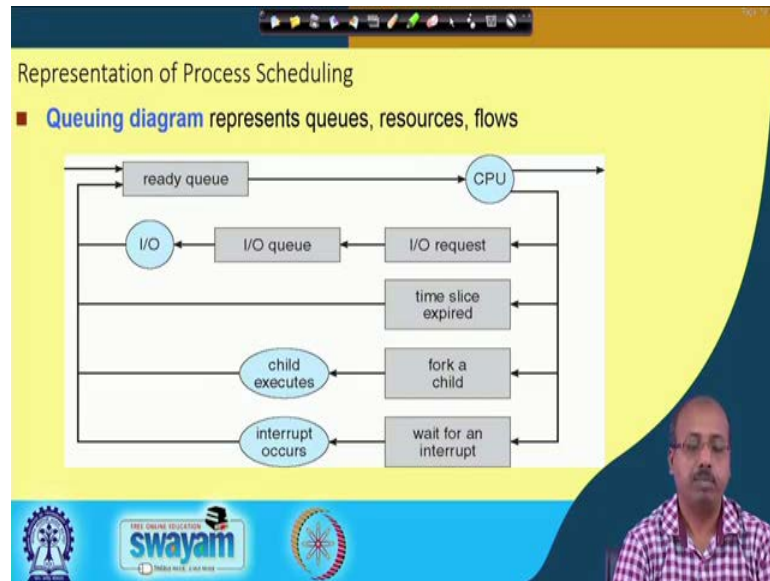


So this is the ready queue and various I O device queue. So ready queue we have got this PCB 7 and PCB 2. So, these are the two processes that are there on that are now ready; out of that this PCB 7 so that process 7 is the first process in the queue and this is a PCB 2 is the second process in the queue, so, they are there.

Now, maybe we have got magnetic; for the magnetic tape device there is no such processes which are waiting for this magnetic tape, the tape unit 0, magnetic tape unit 1

also that may be nothing. Then the disk unit 0 so, we may have got a number of processes which are waiting for this disk like process 3, process 14 and process 6, and for the terminal access may be process 5 is in the queue for doing this things. So, this way we can have at any point of time, so if you take a snapshot of the system then they device a different queues may have different processes waiting in the queue.

(Refer Slide Time: 20:47)



So, a typical queuing diagram that represents the process scheduling is like this. So, initially of processes in the ready queue when the process has been copied into the main memory, so it joins the ready queue. From the ready queue the eventually the scheduling will be, so this particular arrow means that the process has been scheduled. So, it gets the CPU.

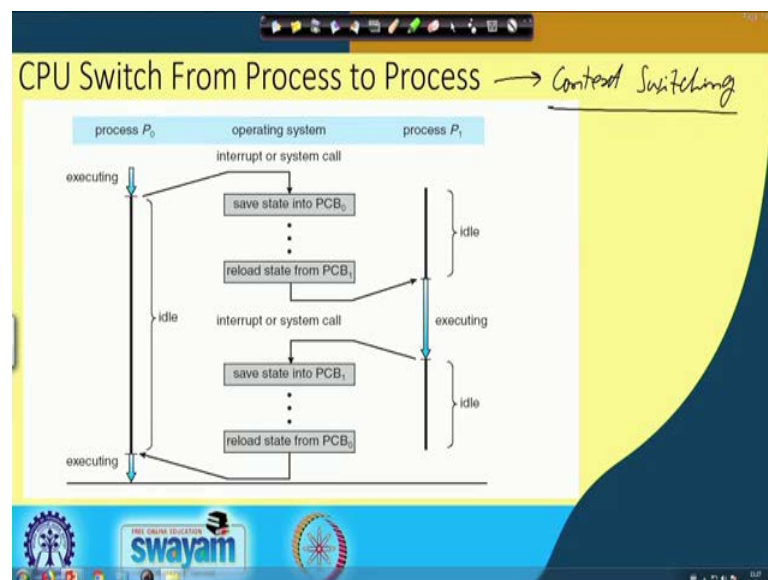
So, when it gets the CPU, so it will be the executing for some time and it may so happen that the process gets over and it goes out of CPU and that is terminated. Or it may so happen that process has come across different situations, like it may request for person I O operation, so in that case it joins the I O queue.

So, this is one possibility another possibility is that the time slice has expired? So, the time slice expires then the process is taken out of CPU and put on to the ready queue, so this is the other operation. Then what happens; so maybe one process when it is executing it creates another process and then it waits for the execution of the child process.

So, this is a typical situation that we have got in this command line interpreters that we have in any operating system. So, if you give a command for execution, then the cell program that we have. So, it has to wait for the (Refer Time: 22:20) for the child to be over. So, this child executes and so after the child finishes, so this process joins on to the ready queue; so, this is one possibility.

Then we can have this interrupt. So, there may be and waiting for some interrupt to occur and when that interrupt occurs it goes back to the ready queue. So, this way we can have different type of situations in which of the processes may be executed in some sequence and accordingly they go through different queues and different scheduling stages in the system.

(Refer Slide Time: 22:56)



So, this is a typical example how this CPU will switch from one process to another process. So, this is known as context switching; so this is known as context switching. So, what happens is that this is the one process that is there so, process P_0 [vocalized-noise, so it was executing. So process P_0 was executing so, this is by particular snapshot of the system and then some interrupt or system call has occurred.

So, it goes to the kernel mode in the operating system though it will save the state of this process 0 into PCB 0. And then process P_1 . So, now the scheduler will come and it will decide which process to be loaded. So, suppose it decides at the process 1 will be loaded.

So, it will reload the state from PCB 1 into the CPU registers, program counters etcetera and then the process 1 is now ready for execution.

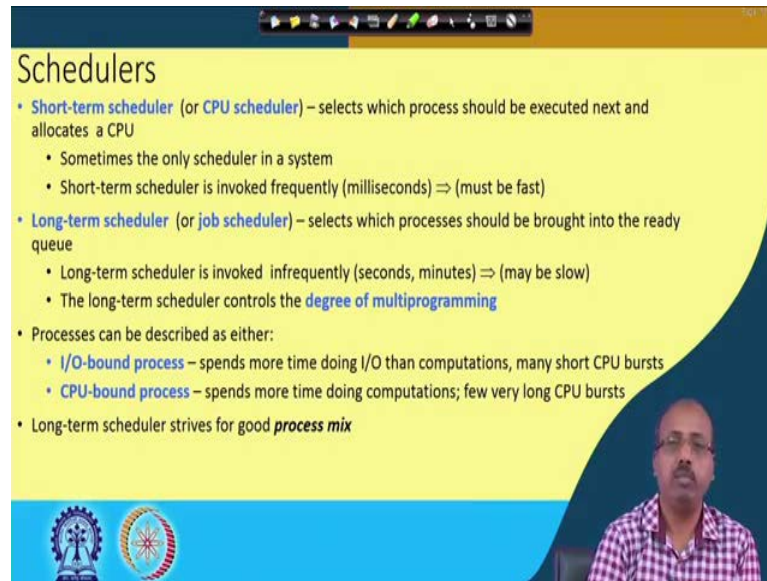
So, the control goes to process 1 and it starts executing, after sometime after the process one has executed. So, again it may be process 1 will generate and interrupt or system call and then when it happens, so it has to save information into the PCB 1 corresponding to process 1. Then after sometime, so this now the scheduler may decide that I have to restart process 0. So, it has to that decision is taken in this part in this region of time and then it was a restart process 0. So, it has to reload state from PCB 0 and then it can give the control back to process 0 and it starts executing from this point.

So, this way we can have switching between processes. So, whenever a context switch occurs. So, this the process which was executing so far its content has to be saved into the corresponding PCB, then the context from then the next processes PCB has to be loaded into the CPU registers and all and then the next process can start.

So, this is known as the context switching now you can understand that this is an overhead this is coming to the system, this saving the context and reloading the context, so that is basically the context switching and that takes the time. And so effectively, so you can say that this is and wastage of CPU time; so, we will try to minimize this context switching to as little a value as possible and accordingly the CPU usage will be going up.

But definitely there is a question like if you just reduce it to some to a arbitrary value, then it becomes more of a first come first of type of situation and that way the overall time maybe that we will see that it has got some bad implication in the system performance. So, that creates the difficulty.

(Refer Slide Time: 26:14)



Schedulers

- **Short-term scheduler** (or **CPU scheduler**) – selects which process should be executed next and allocates a CPU
 - Sometimes the only scheduler in a system
 - Short-term scheduler is invoked frequently (milliseconds) ⇒ (must be fast)
- **Long-term scheduler** (or **job scheduler**) – selects which processes should be brought into the ready queue
 - Long-term scheduler is invoked infrequently (seconds, minutes) ⇒ (may be slow)
 - The long-term scheduler controls the **degree of multiprogramming**
- Processes can be described as either:
 - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
 - **CPU-bound process** – spends more time doing computations; few very long CPU bursts
- Long-term scheduler strives for good **process mix**

The slide features a yellow background with a blue footer containing two logos. A video inset in the bottom right corner shows a man with glasses and a checkered shirt speaking.

So we will be talking about different types of schedulers which will take a decision like which job to be executed next, when the ready queue there are a number of processes which are waiting for this jobs to be selected for execution. So, they are that is done by short term scheduler. Then we have got long term scheduler which will be selecting the jobs that should be executing that should be made ready for execution, so that is the long term scheduler.

And sometimes we have got a midterm scheduler also which is not there in all operating systems, what happens is that there may be a number of processes which have completed their I O operations. So, out of that some of the processes they should be they are put back on to the ready queue. So, that way we can have this midterm scheduler also. So, we will look into the scheduler operations in the next class.