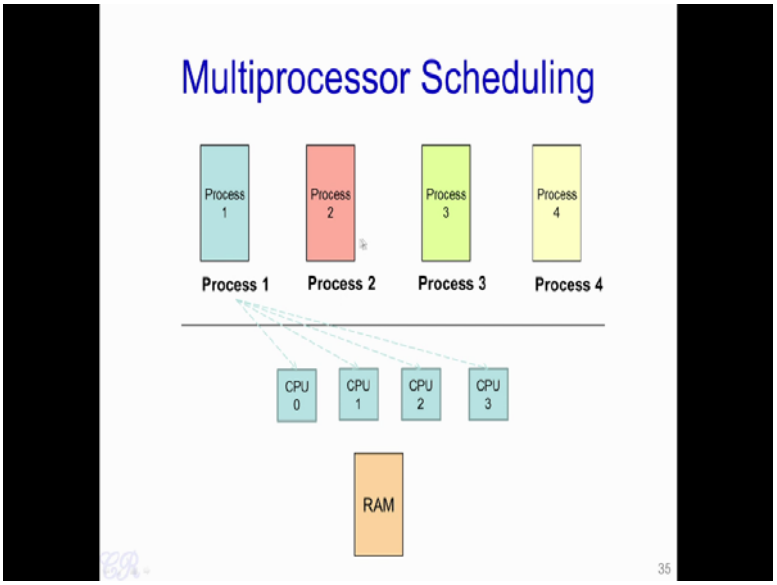
Introduction to Operating Systems
Prof. Chester Rebeiro
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Week - 05
Lecture – 20
Multiprocessor Scheduling

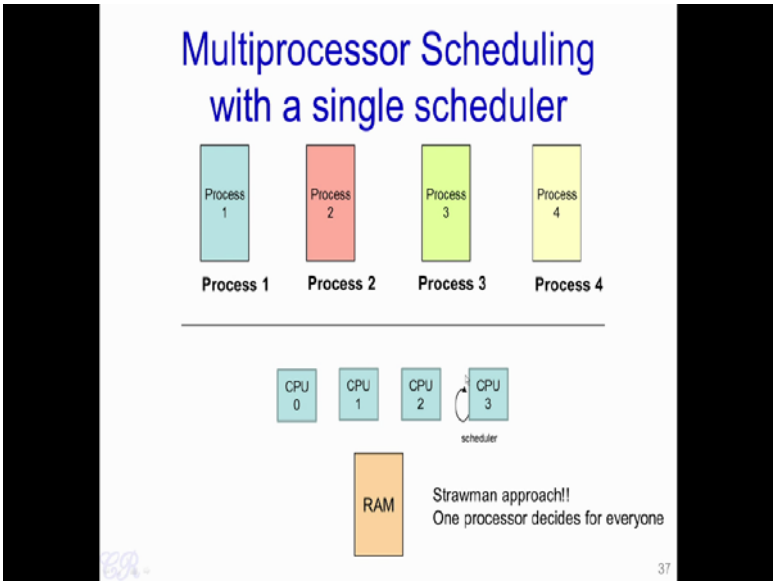
Hello. So far we have seen how scheduling algorithms are designed for a single CPU. So, the scheduling algorithm would choose a process to execute for the CPU. In this particular video, we will look at Multiprocessor Scheduling Algorithms.

(Refer Slide Time: 00:35)



Essentially, we will see that if we have multiple processors in the system, how a scheduling algorithm could schedule processors into these various CPU's.

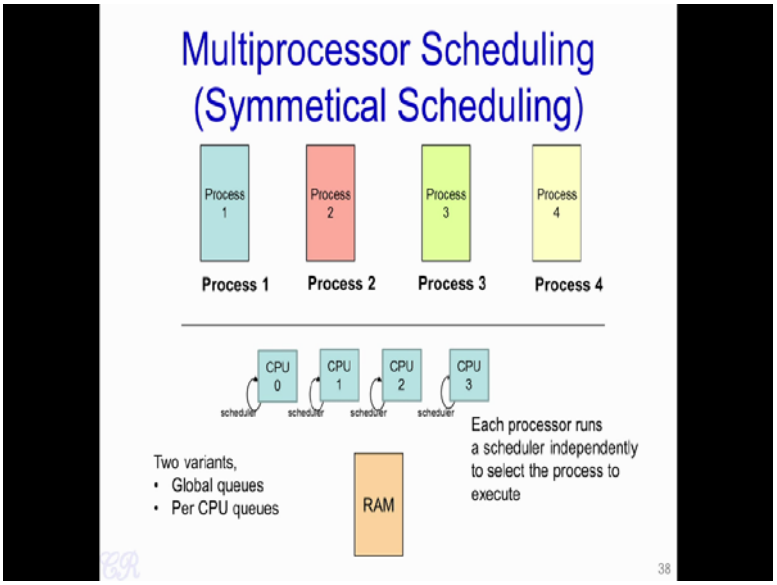
(Refer Slide Time: 00:45)



One very simple scheme for a multiprocessor scheduler is where there is a dedicated CPU to run the scheduler. This scheduler decides for everyone, essentially decides which process should run in which CPU. Now implementing this scheme is very simple. So, this scheduler is going to have local queue of processor which are ready to run and it would use some mechanism to decide which of these processors should be scheduled into which CPU. The limitation, as one would expect is the performance degradation.

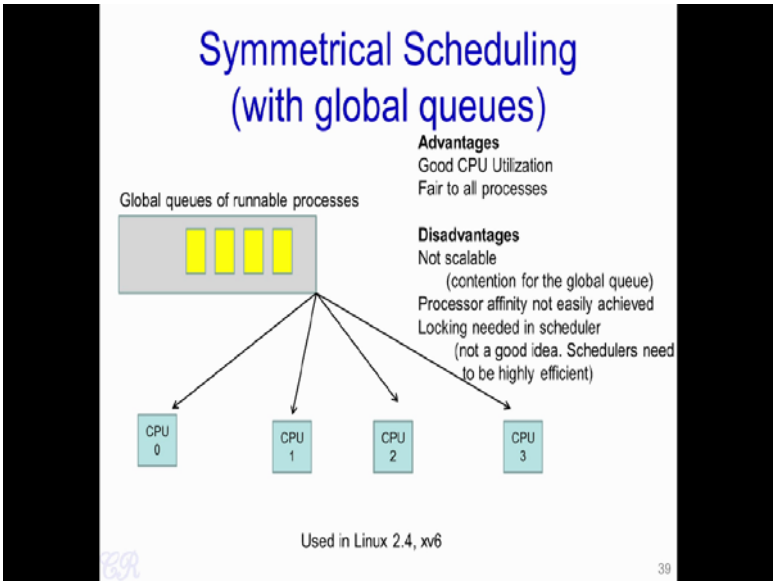
Since, all CPU's are waiting for the scheduling CPU to tell it which process to execute next and this could happen for instance every 10 milliseconds therefore, there could be significant over heads due to scheduling.

(Refer Slide Time: 01:37)



Another multiprocessor scheduling scheme is the symmetrical scheduling scheme. Here instead of a single CPU which decides for all processors in the system, here each CPU runs its own scheduler which is independent of each other. Therefore, each CPU at the end of a time slice decides which process it is going to execute next. Now, there are two variants of the symmetrical scheduling scheme - one is with a Global queue another with Per CPU queues. We will look at each of these variants next.

(Refer Slide Time: 02:16)



In the symmetrical scheduling scheme with global queues, there is exactly one queue of ready processes which is shared among all processors. These schedulers in each CPU would need to look up this global queue to decide upon which process to run next.

For instance, CPU 0 will need to look up this global queue and decide on a particular process to run next. That process will then change from the ready state to the running state. The advantage of the scheme is that there is good CPU utilization and fairness to all processors. The drawback of this particular scheme comes from the fact that a single queue is shared among the various processors in the system. Thus, we could reach a state where there are two processors which query the queue at exactly the same time and pick exactly the same process to execute. Thus, a single process may execute in the same instant of time in two different CPU's.

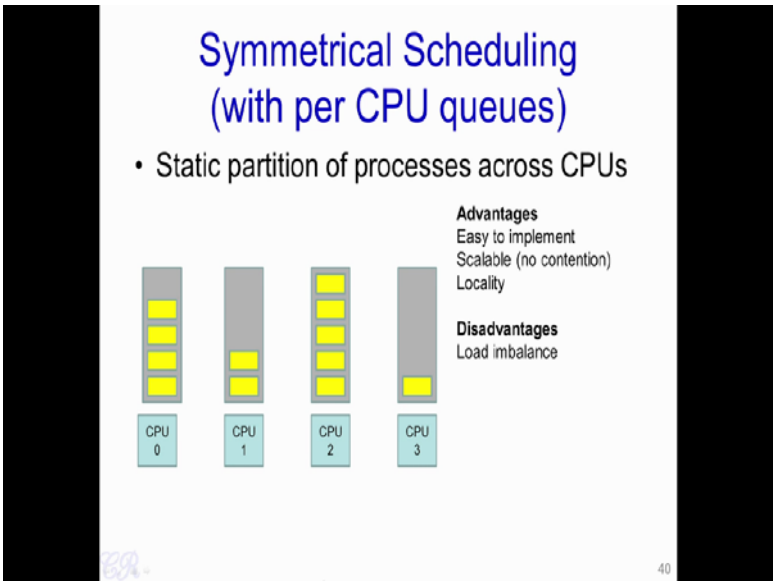
In order to prevent such issues it is require that access to this particular queue is serialized. That is, if CPU 0 wants to access the queue no other CPU should be able to access that queue during that particular time. This ensures that only CPU could choose a process at a particular instant of time.

Now, this mechanism of serializing access to the global queue is achieved by a technique

known as locking. Now, while this locking mechanism would work it is not a good idea because it will not scale. Essentially, consider the fact that instead of 4 processors in the system there are now 64 processors and all these processors have a serialized access to this global queue. Thus, when CPU 0 is accessing this global queue all other processors will have to wait and this could lead to considerable amount of time during scheduling.

Another disadvantage comes from the fact that processor affinity is not easily achieved. So, what is Processor Affinity? Processor Affinity is an option given to users in the system to choose which processor they want their process to execute in. For instance, user may decide that he wants his process to execute only in CPU 0 and no other CPU's. Now, in this particular scheme it is more difficult to implement processor affinity. Now this particular scheme is used in Linux 2.4 Kernels as well as the xv6 operating system.

(Refer Slide Time: 05:12)



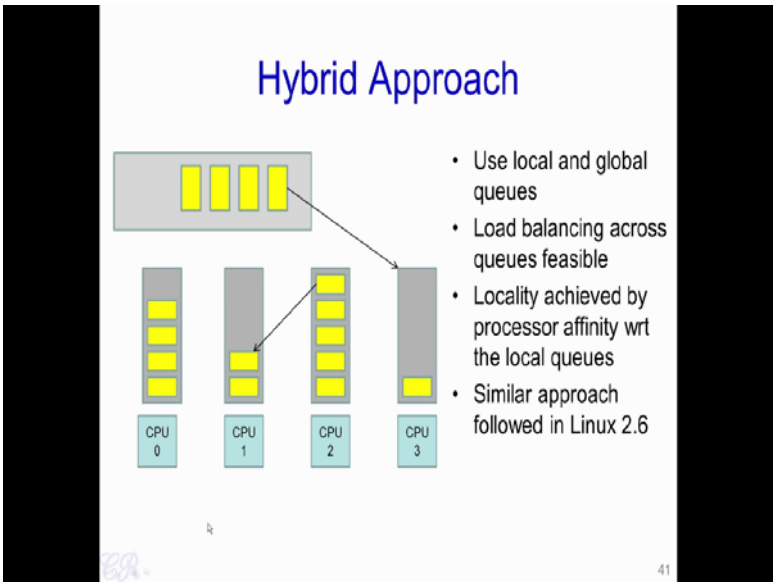
The second variant of symmetrical scheduling is where there is a single CPU queue for every CPU that each CPU has its own queue of ready processors. Thus, each CPU need to only look at its own queue to decide which process it needs to execute next.

Now, this particular strategy uses a static partitioning of the processes that is at the start of execution of a process either the user or the operating system decides which CPU the

process needs to execute, and the process is then placed in that corresponding CPU's queue. As you can see, the advantage is that it is easy to implement, there is no locking mechanism present, so it is quite scalable as well as due to locality in the sense that each CPU just needs to look at its own CPU queue and is not bothered about any other CPU queues that are present. Therefore, there is no degradation of performance. The advantage of the scheme is that it is easy to implement and it is scalable. Scalable comes from the fact that there is no locking mechanism or serialization of the accesses to the CPU queues.

Another advantage is that the choice is local. Essentially the CPU 0 is only concerned with its own queue and not concerned with any other queue present in the system. Therefore, the choice made is very quick, there is minimum performance degradation. The drawback of the scheme that it would lead to load imbalance, load imbalance occurs when some CPU queues have a lot of processes to execute or rather some CPUs have a lot of processes in the ready state while other processes such as CPU 3 have just very few processes in the ready state. Thus CPU 2 is doing lot more work compare to CPU 3.

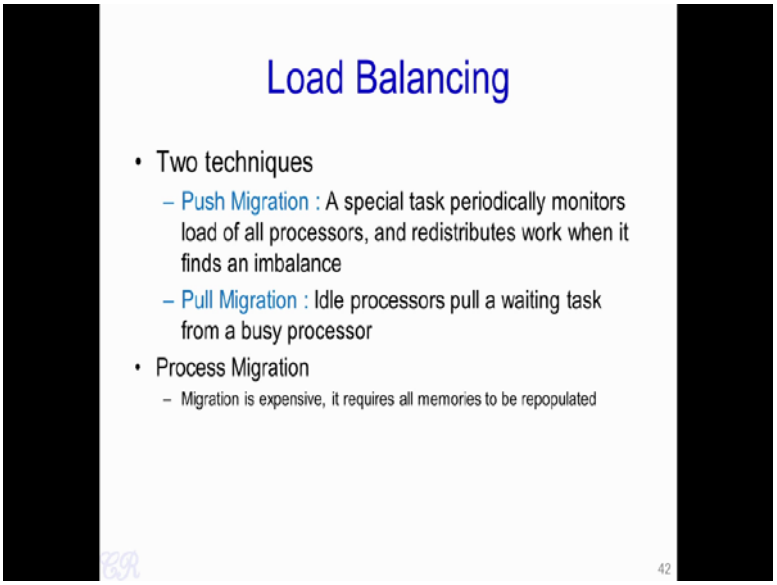
(Refer Slide Time: 07:29)



A third way to do symmetrical scheduling is a Hybrid Approach. So, this approach is used in Linux Kernels from 2.6 onwards. So, essentially this approach uses both local

queues as well as global queues. The local queues are the queues which are associated which each CPU. Essentially, each CPU has its own local queue while the global queue is shared among the CPU's. The global queue is used to ensure that load balancing is maintained that is it is going to ensure that each CPU would have a fair share of the processes to execute. The local queues would ensure locality, by having the local queue the performance degradation of the system is minimized.

(Refer Slide Time: 08:23)



Beside the global queue there are two more techniques to achieve Load Balancing - one is the Push Migration and the other is the Pull Migration. With the Push Migration, a special task would periodically monitor the load of all processors and redistribute work whenever it finds an imbalance among the processors. With Pull Migration, an idle processor will pull a task from a busy processor and start to execute it.

Thus migrating processors from busy processors to less busy processors will achieve load balancing. This being said process migration should be done with the pinch of salt. Essentially, migrating a process from one CPU to another is expensive. Whenever a process migrates from one CPU to another, all memories with related to the new CPU needs to be repopulated, by this we mean the cache memory associated with that CPU needs to load the migrating processors instructions and data. Similarly, the TLB needs to

be flushed and so on. This could lead to significant over heads. Thus process migration should be done with only if required.

With this we will end this video which looked at Multiprocessor Scheduling Schemes and Load Balancing among the different CPU's.

Thank you.