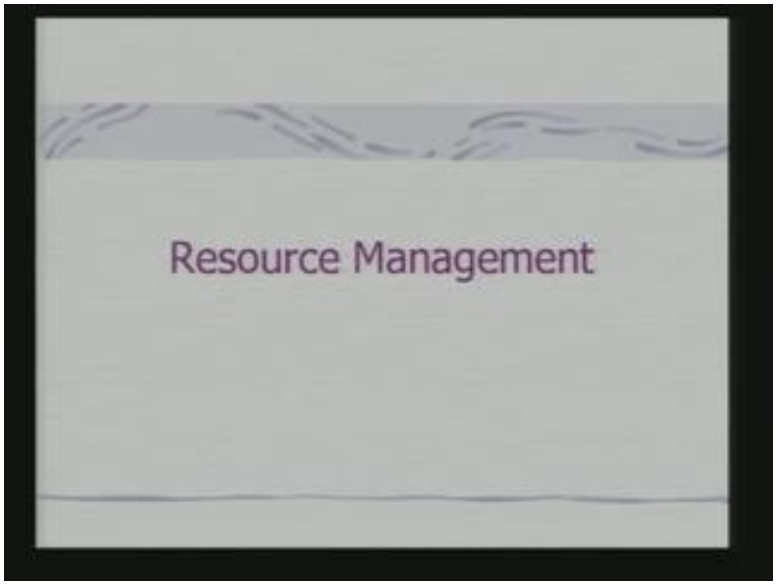
Embedded Systems
Dr. Santanu Chaudhury
Department of Electrical Engineering
Indian Institute of Technology, Delhi

Lecture - 22
Resource Management

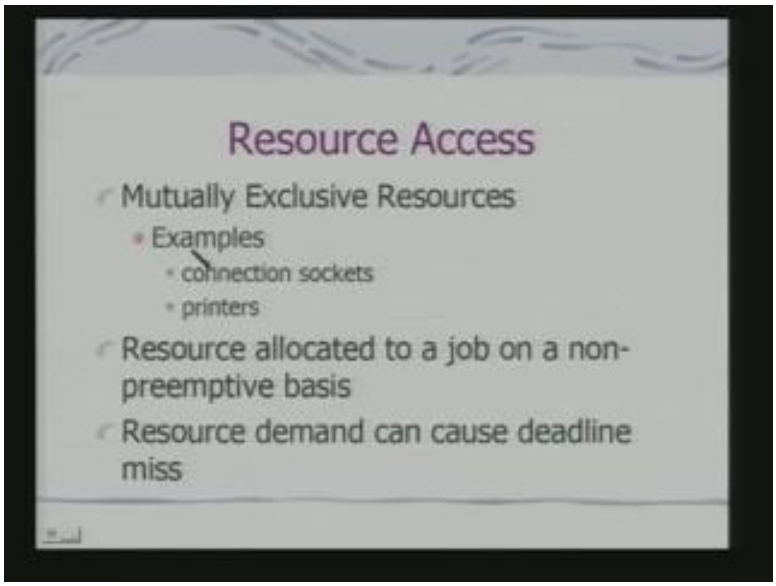
In the last class, we had discussed the scheduling policies for a real time operating system and we have found that we need to modify these policies on need to relook at these policies. When there are processors sharing resources and there are conflicts regarding the demand of resources between these processes. Today, we shall look at resource management.

(Refer Slide Time: 1:45)



Resource management involves scheduling for resources, as well as strategies to be followed for special resources; other than the CPU like memory as well as IO devices. So, let us recapitulate what we did in the last class about the resource access.

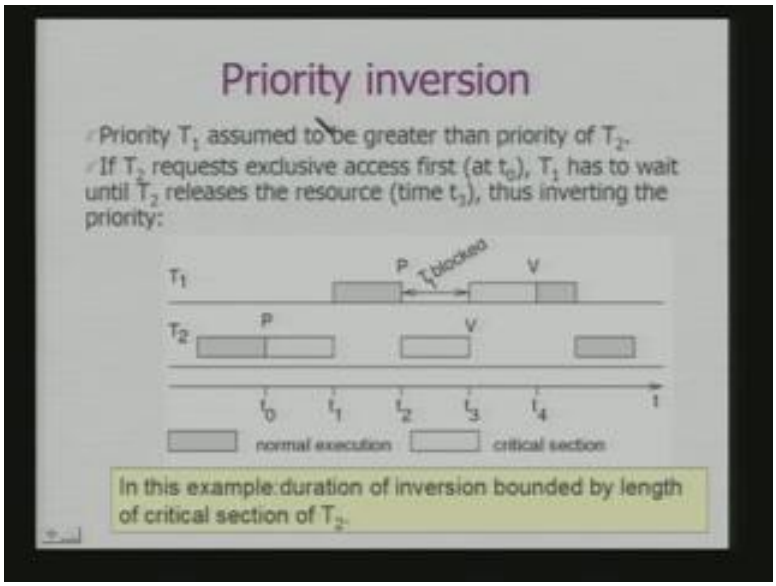
(Refer Slide Time: 2:08)



Many of the resources are actually mutually exclusive resources. Example can be printers, examples can be sockets by which, a system establishes network connection. These resources are allocated to a job on a non-preemptive basis. That means a process gets an exclusive access to the resource. And this resource demand and the conflict due to resource demand can actually cause deadline mix.

And also priorities of the processes that have been assigned, depending on the periodicy or the deadline may not be valid and may not be schedulable. According to these priorities; because of these resources demands so, let us at look at this problem of what we called priority inversion.

(Refer Slide Time: 3:05)

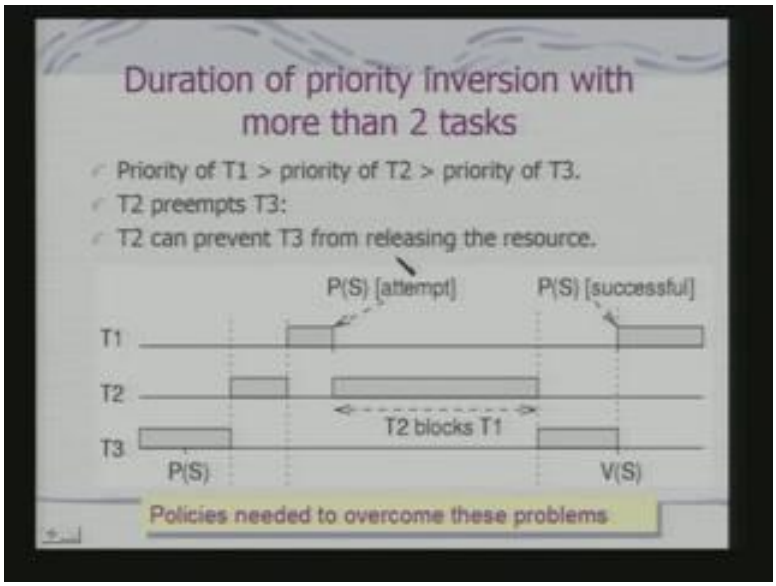


In this case, the priority of the task T 1 is assumed to be greater than that of T 2. Now, let us consider that T 2 asks for a resource at this point. So, we use the P and V operations for gaining exclusive access to the resource. So, when it executes P operation the task T 2. Since, no other task is using that resource, it goes in to what is called a critical section that is the section in which, it uses an exclusive access.

So, goes in to the critical section, but now next what happens T 1 at this point gets scheduled, T 1 starts execution. But, the resource is still with T 2 because it has not yet executed V that is release operation. So, when this task T 1 asks for the resource it gets blocked. And it waits, so it slips and while it is slipping the T 2 really becomes able to be executed. So, T 2 gets scheduled and it is now being executed. So, in this case effectively although the T 1s job is not yet complete. The T 2 gets scheduled even though it has got a lower priority.

So, this is an example of priority inversion. And the duration of inversion is bounded by the length of the critical section of T 2. So, when the critical section is over, it is released at V then the T 1 gets the resource. It executes in the critical section then releases the resource at V that it continues to be scheduled; because the job is not yet finished. And then only after its task is done T 2 get scheduled. So, this is according to the normal priority. So, what we find that when we require an exclusive access to resource, the priorities can actually get inverted.

(Refer Slide Time: 5:17)



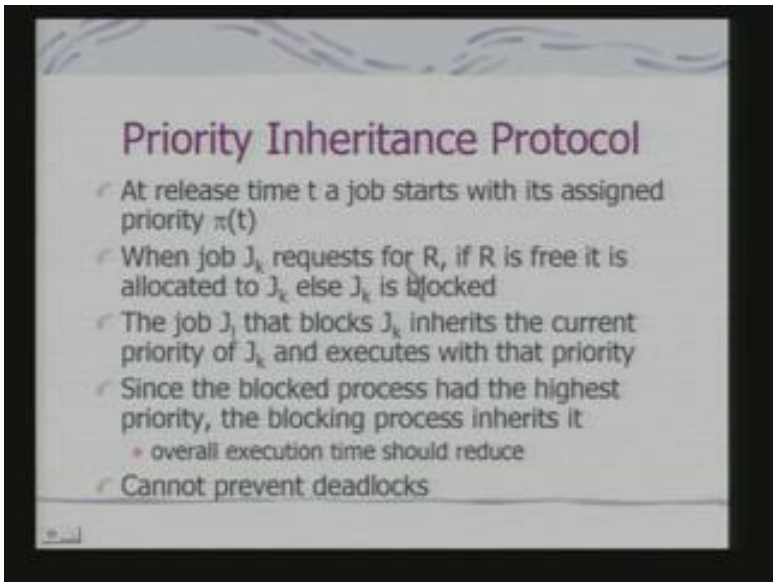
The problem becomes more complex. If we have a number of processors demanding the resource or if there are other processes as well in the system, which may not be actually using the resource. So, here what happens is that priority of T 1 we have considering a situation, where priority of T 1 is greater than priority of T 2 and T 2s priority is greater than that of T 3. So, at this point the T 2 is in execution and then the T 1, since it priority T 1 comes in here.

But, what happens is there is a P S that there is an attempt to get the resource the P operation. But, that becomes blocked because T 2 can already have the resource with the T 2 having the resource the T 1 is blocked, so T 2 starts executive. So, here there is a priority inversion. But, after that the T 2 blocks and T 2 executes, but who is having the resource, the resource is actually with T 3 because it has successfully executed the P operation.

So, the resource is with T 3, T 1 is blocked. So, T 2 gets executed, but T 2 finishes it is execution, the control that is the CPU is not schedule with T 1, but it is schedule with T 3, because T 3 is having the resource and T 1 is blocked for the resource to be released by T 3. So, now T 3 starts execution. So, once T 3 finishes then only T 1 gets a time to execute. So, what did appears is that when, there is a mix of such processors then inversion period can be even more than that of the critical section.

Because here what has happened is T 2, since it has a higher priority than that of T 3. And since it does not require the resource, it gets scheduled and T 3s critical section period gets stretched because it is not getting the CPU. So, effectively T 1, which is having a highest priority, gets blocked for a longer period of time. So, this is really a problem, when your higher priority trade of the task is handling critical tasks. So, there has to be policies to overcome these problems.

(Refer Slide Time: 7:57)

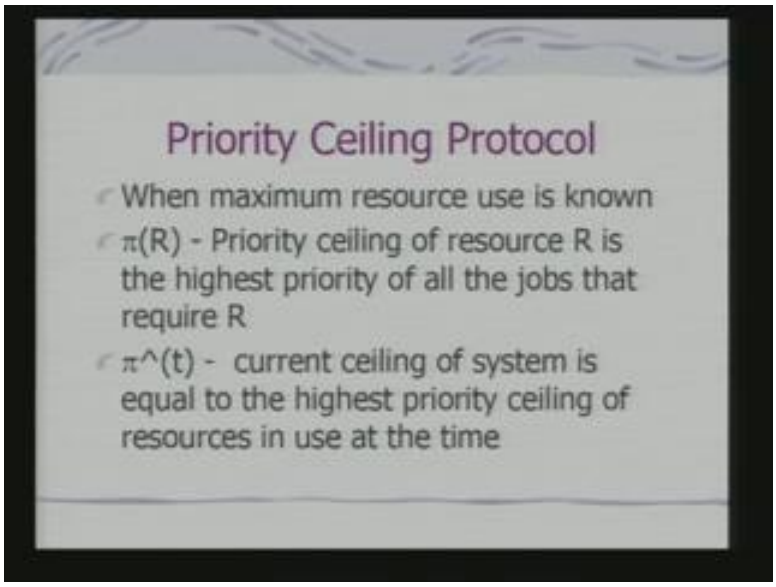


So, the most simple, the simplest policy that we really have is what is called priority inheritance protocol. In priority inheritance protocol, what happens is when a job Jk request for a resource R, if R is free it is allocated to Jk else Jk is blocked have these step is standard, which happens on the normal conditions. But, something additional happens in case of priority inheritance protocol the job Jl, which is blocking Jk, now inherits priority of Jk and executes with that priority that means in the previous example, T 1 was blocked because T 3 had the resource, T 3 was having minimum priority.

But because of priority inheritance, now T 3 will having, the priority same as that of T 1 that means, T 3 priority will be more than that of T 2, so it will finish of its critical section earlier. So, what we say since the blocked process had the highest priority, the blocking process inherits it. So, over all execution time should be reduced. So, the period for which actually the priority inversion takes place would be small enough.

But, obviously this cannot prevent deadlocks, when can deadlock happen, when processes are looking for multiple resources. And if process and the task T 1 has resource R 1 and is looking for resource R 2 and the process of the task T 2 have the resource R 2 and is looking for R 1 there would be a deadlock. Even priority inheritance cannot eliminate deadlock situation. There is another protocol which is better, because it is manage things in a more appropriate fashion is priority ceiling protocol.

(Refer Slide Time: 10:00)

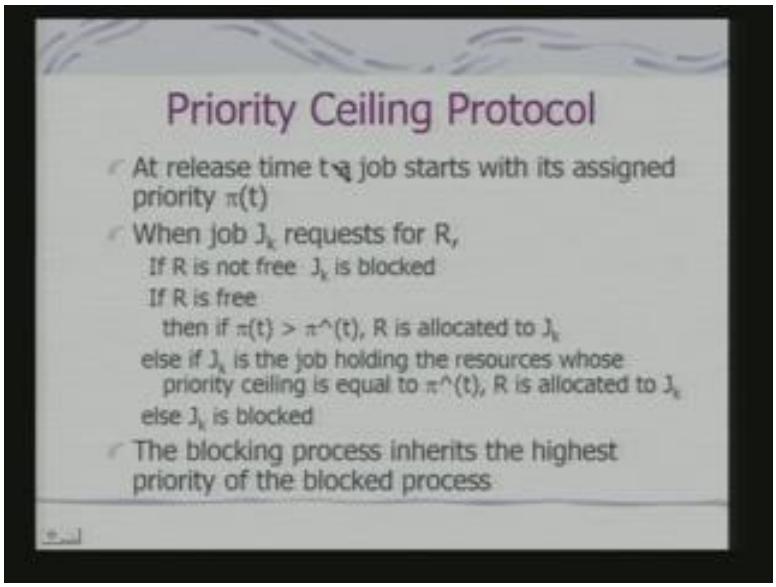


Now, here what is the assumption, the maximum resource use is known. So, in fact I know the maximum resource usage, there are algorithms and techniques by which I can schedule the resources, so that deadlock do not occur. So, that means I can prevent deadlock. So, under similar assumptions priority ceiling protocol works that means it assumes that maximum resource use is known.

And it introduces two entities, two quantities pie R and pie hat t, what is pie R, pie R is the priority ceiling of resource R and it is the highest priority of all the jobs that require R, because since you know the resource use by each and every task. So, you can compute pie R because this is the priority of all the jobs that require resource R. And pie hat t is a current ceiling of system; it is equal to the highest priority ceiling of resources in use at the time.

That means, there may be a set of resources, which have been used and that priority ceiling or the current ceiling will be equal to the priority of the, maximum priority of the resources which are currently used, then what is done.

(Refer Slide Time: 11:35)



At release time t jobs starts with its assigned priority, because this is the priority define by the scheduler, when a job Jk request for R that is the resource R, if R is not free Jk is; obviously, blocked. But if R is free, R is not straight away allocated what is done, what is checked is if pie t is greater than pie hat t, pie hat t is the priority ceiling pie hat t then R is allocated to Jk else if the Jk is the job holding the resources was priority ceiling is equal to pie hat t then r is allocated to Jk else Jk is blocked.

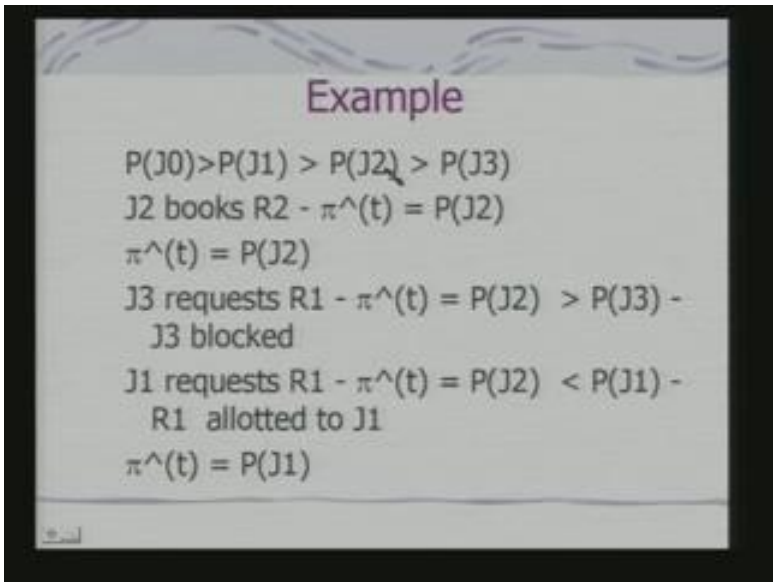
What is the motivation of this, motivation is the job is having higher priority than that of the priority ceiling; that means, it is more important than the jobs which are currently using different resources. So, the resource should be granted to the requesting job, the other scenario is that current priority ceiling may be because of the usage of the resource by the current process itself. Although its current priority may be less or equal to that of pie hat t, it should be equal to pie hat t.

So, under that condition, it should have the resource because if it has the resource then it can finish of the job and can release resource for others, otherwise it will continue to get the continue to way to get the resource. So, the resource get allocated to Jk and the

blocking processes inherits the highest priority of the block process, if there are multiple such block process its get a highest priority, this is same as that of priority inheritance.

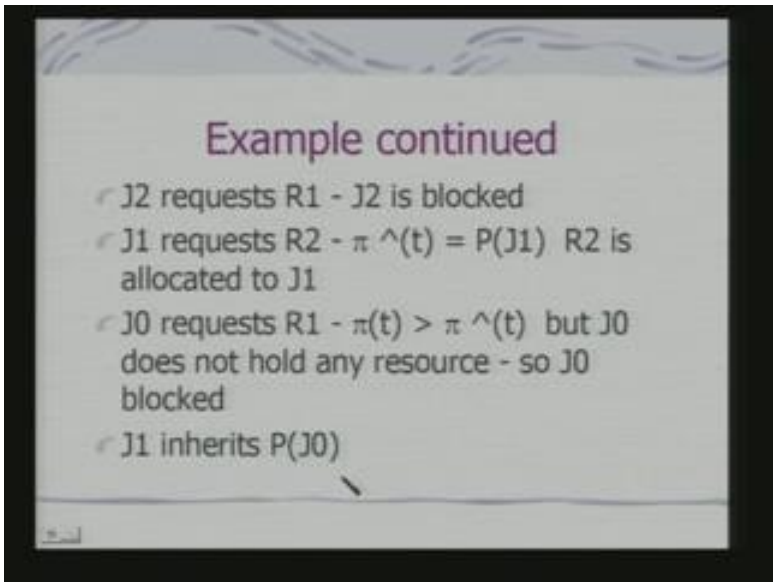
So, therefore priority ceiling protocol is an improvement up on your priority inheritance protocol. What is implication here if you look in to it, the basic implication is that depending on the priority of the resources, which I had in use the resources are getting allocated, resources are not just allocated on the basis of their availability. So, what is being ensured, apprises with the higher priority will get the resource and will finish of its job earlier than other processes. So, let us take an example to understand this. So, I have got these jobs.

(Refer Slide Time: 14:10)



J 0 J 1 J 2 J 3 and the priority of J 0 is greater than that of J 1 greater than J 2 and that of J 3. Now, J 2 books R 2, so what will be the pie hat t, R 2 is the resource will be that of J 2, priority of J 2 and J 3 requests R 1. So, pie hat t now is greater than that of J 3. So, J 3 will get blocked. Now, J 1 requests R 1, now pie hat t is that of J 2 which is less than that of priority of J 1, so R 1 should be allocated to J 1 and now the priority ceiling becomes that of process J 1.

(Refer Slide Time: 14:59)



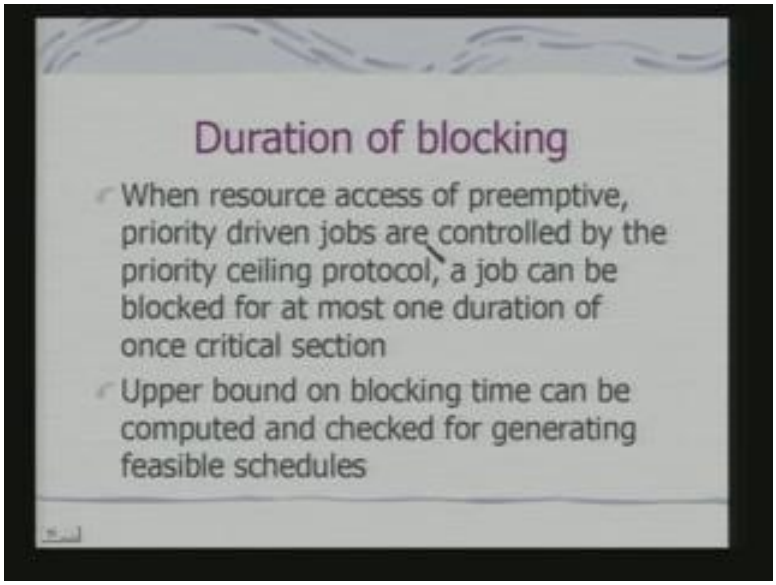
So, J 2 now request R 1, but J 2 will be blocked because currently its priority less than that of the priority ceiling. And J 1 requests R 2, so pie hat t is equal to priority of J 1, so R 2 will be allocated to J 1. Because, understand that If I would have allocated here, the J 2 is not possible to complete its jobs because it is requesting R 1. And J 1 is requesting R 2, so it is getting all the resources because it is of the high priority. And J 0 request R 1, when J 0 request R 1, this pie t is greater than pie hat t, but J 0 does not hold any resource, so J 0 is blocked.

So, this is a current ((Refer Time: 15:51)) which is there in the priority ceiling protocol, it enables a process which has got the resources to finish of its task. So, the resource does not get allocated to a process because of it higher priority only, if I would have allocated that this might would have lead to a deadlock situation. And J 1 inherits priority of J naught because J 1 is blocking J naught, so J 1s priority increases, so it is likely to finish of faster.

So, this is how you manage the resources based on the priority associated with the processors, when the processors are competing for the shared resources and resources need to be allocated as an exclusive resource. In fact, this is a very critical problem in variety of embedded systems, because if the blocking is larger than there may be critical deadlines can be missed and accordingly there could be bug of the falls plugging the

system behavior, it may actually detrimental effect the system behavior. So, how do you calculate the duration of blocking.

(Refer Slide Time: 17:12)



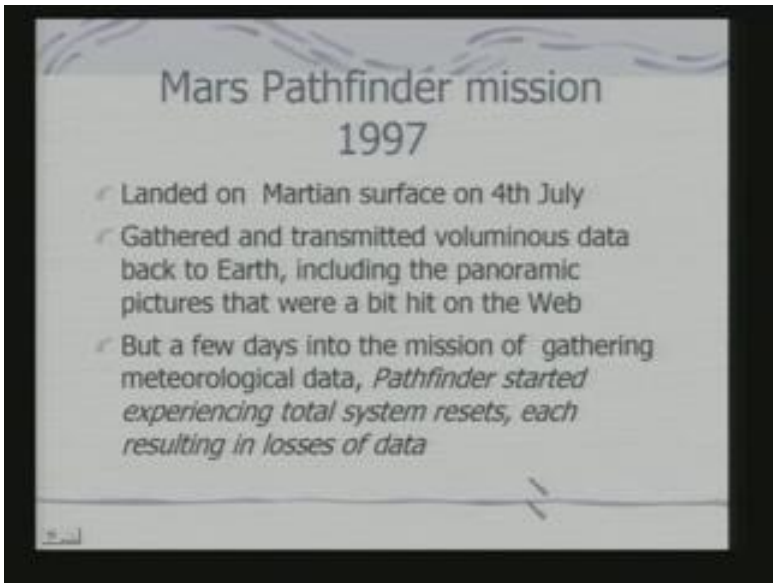
When resource access of preemptive, priority driven jobs are controlled by the priority ceiling protocol, a job can be blocked for at most one duration of one critical section. And upper bound on blocking time can be computed and checked for generating feasible schedules because you exactly know why this is possible, this is possible simply because you know the resource usage before hand.

So, you know the priorities to be associated with the resources and why the first condition becomes true. When resource access of preemptive, priority driven jobs are controlled by the priority ceiling protocol, a job can be blocked for at most one duration of one critical section. It cannot be block for more because the resource is getting allocated to the job, which is having the priority ceiling and it is having other resources as well. And it will not get blocked because of the processor being allocated to another process because it will inherit the priority of the highest blocked job. So, these are the advantages of priority ceiling protocol and in fact this is better than that of your priority inheritance protocol.

(Refer Slide Time: 18:28)

So, we shall look at a case study to understand this priority inheritance and then the priority inversion, how it creates problem and this is a real life problem, which happened with mars rover pathfinder this is…

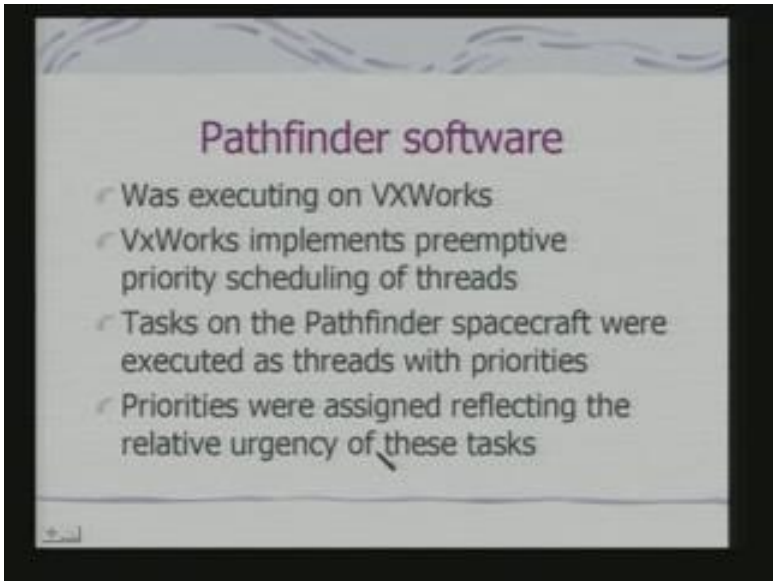
(Refer Slide Time: 18:39)



It was a mission in 1997, so what it says is that it gathered transmitted voluminous data back to earth, including panoramic pictures that were a bit hit on the web, in fact one NASA web site those pictures are even today available. But, a few days into the mission of gathering metrological data, pathfinder started experiencing total system resets, each

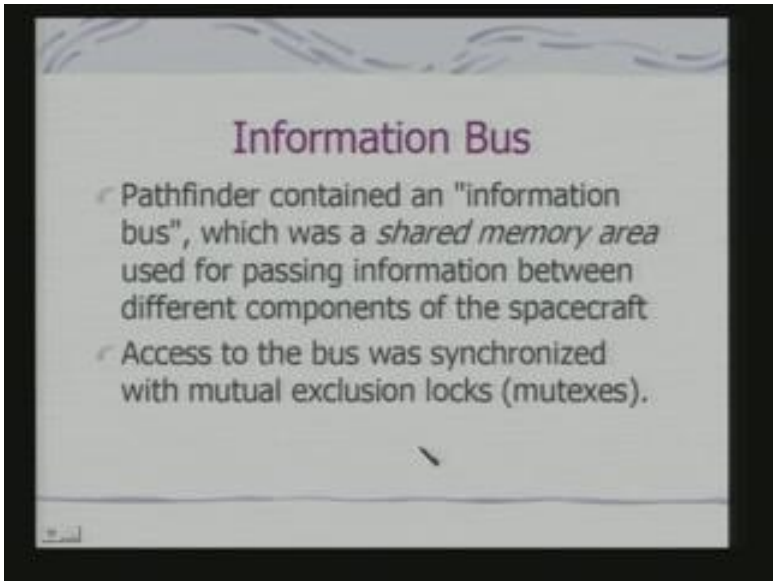
resulting in losses of data because whatever data is gathered, if the system gets reset then there is a loss of data.

(Refer Slide Time: 19:09)



The whole software was working on a operating system was commercially available operating system VxWorks. And VxWorks is an operating system, VxWorks implements preemptive priority scheduling of threads. Tasks on the pathfinder spacecraft were executed as threads with priorities. Priorities were assigned reflecting the relative urgency of these tasks this is the standard way of assigning priorities.

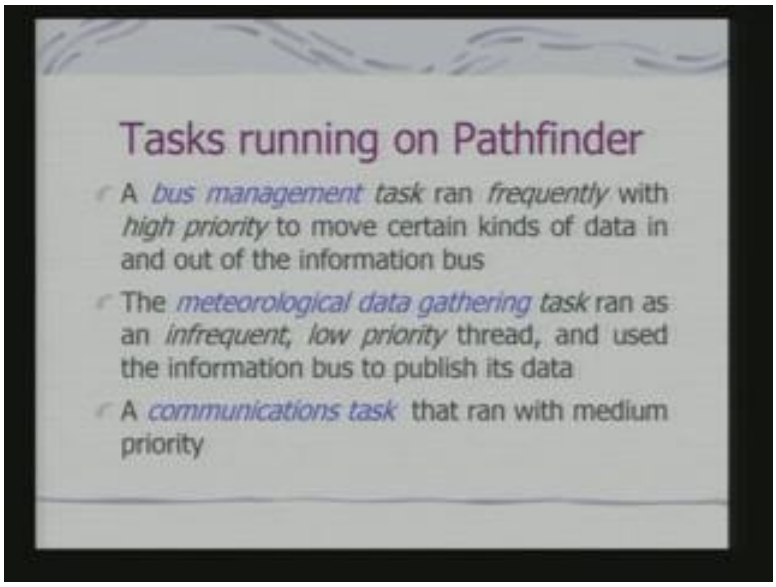
(Refer Slide Time: 19:39)



And why threads are being used, because I told you the threads in an embedded system is the most likely mechanism to manage concurrency because the context switching overhead for threads is much less. So, pathfinder contained what is called an information bus, which was a shared memory area used for passing information between different components of spacecraft. So, basically this processes threads also need mechanism for communication, shared memory is one mechanism for communication between threads as well as that of the process.

So, that shared memory that being referred to as information bus. Obviously, the access to the bus was synchronized with mutual exclusion locks, just like our P N, V that we are using. Because, there cannot be more than one process using the data, which is there in the shared memory area.

(Refer Slide Time: 20:37)

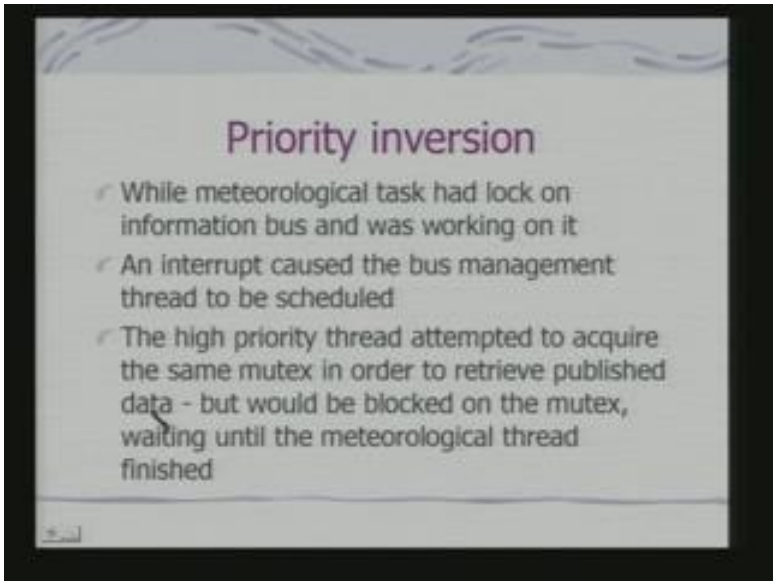


So, we shall look at three basic tasks and how they interacted in terms of their access pattern to the shared memory. A bus management task ran frequently with high priority to move certain kind of data in and out of the information bus, because this is the bus management task, this is basically managing what is so called information bus.

The meteorological data gathering task ran as an infrequent, low priority thread and used the information bus to publish its data. And communication task ran with medium priority, in fact communication task if you communicating meteorological data, it will access the information bus to get the data. So, there are these three basic tasks, these

tasks was high priority task and why it is a critical task and this is a data gathering task and this is a communication task.

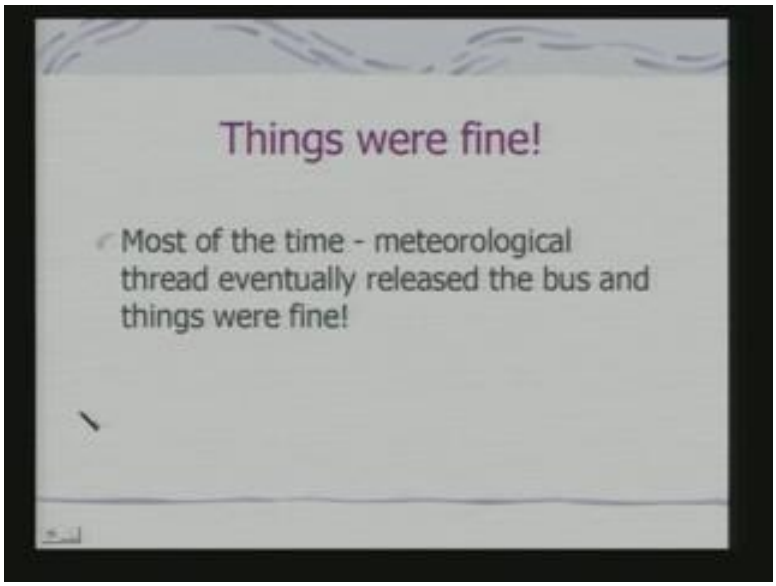
(Refer Slide Time: 21:37)



Now, there is a priority inversion why, while meteorological task had lock on information bus and was working on it. And interrupt caused the bus management thread to be scheduled. Bus management can be there be an interrupt because of and bus management thread is of higher priority. The high priority thread attempted to acquire the same mutex in order to retrieve published data, but would be blocked on the mutex, waiting until the meteorological thread finished, because it is now got an exclusive access to the shared resource.

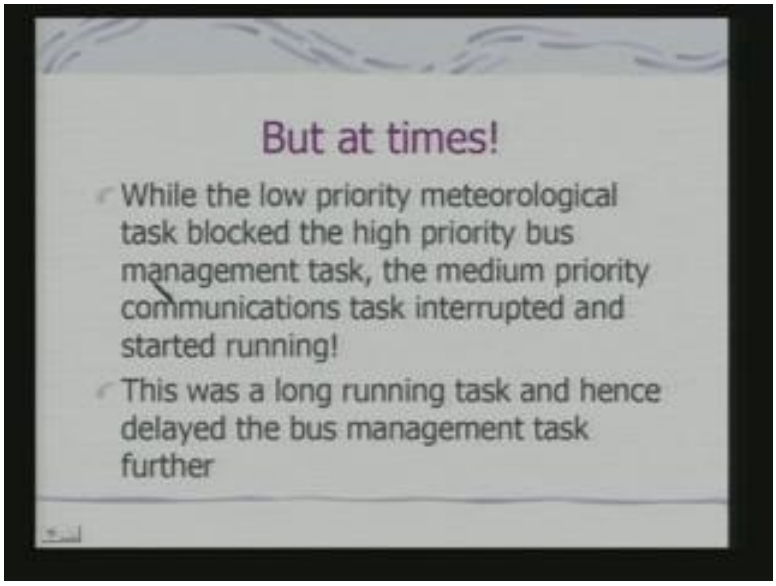
So, now the priority gets inverted.

(Refer Slide Time: 22:14)



But, things were fine most of the time meteorological thread eventually released the bus and things were fine. Because your bus management thread got back the information bus as shared memory and manage the system while. But, at times while the low priority meteorological task blocked the high priority bus management task

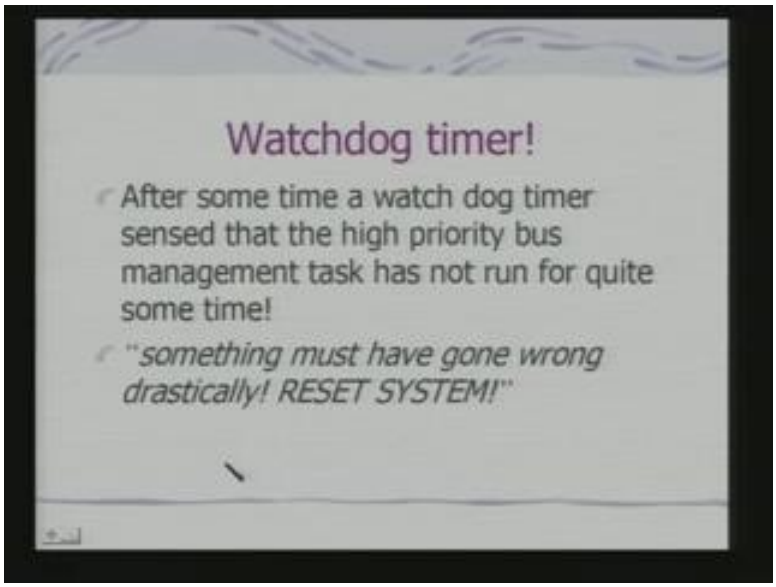
(Refer Slide Time: 22:36)



The medium priority communication tasks interrupted and started running, it can start running and do jobs for which it does not required information bus access. This was a long running task and hence delayed the bus management task further, when as long as it

is not requiring information bus access, it can continue to run. And it is managing basically the communication link with that of the art station. So, what happens every embedded system has got a watchdog timer.

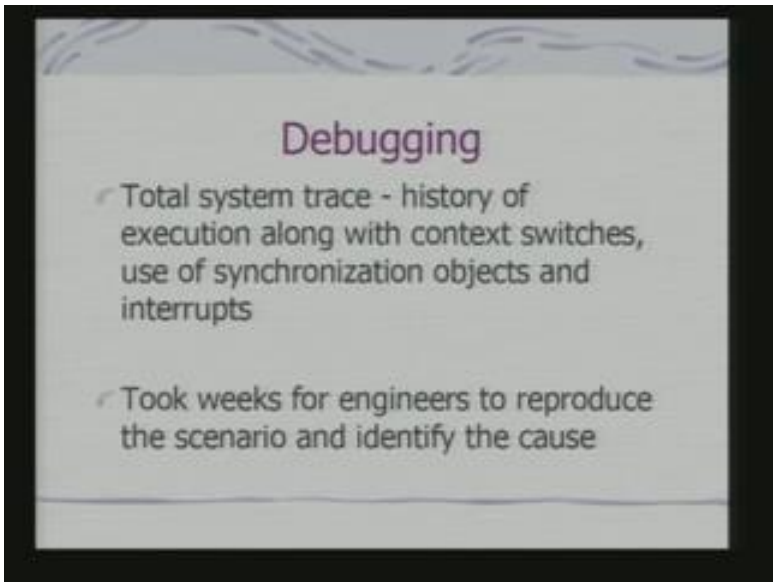
(Refer Slide Time: 23:11)



After sometime a watchdog timer sensed, that the high priority bus management task has not run for quite some time, because if I have program date for a certain time period. Once the time period expires and the high priority task would be given the job of reinitializing the watchdog timer. So, watchdog timer is not getting reinitialized. So; that means, system has gone in to some kind of an unknown state. So, something must have gone wrong drastically and you reset the system.

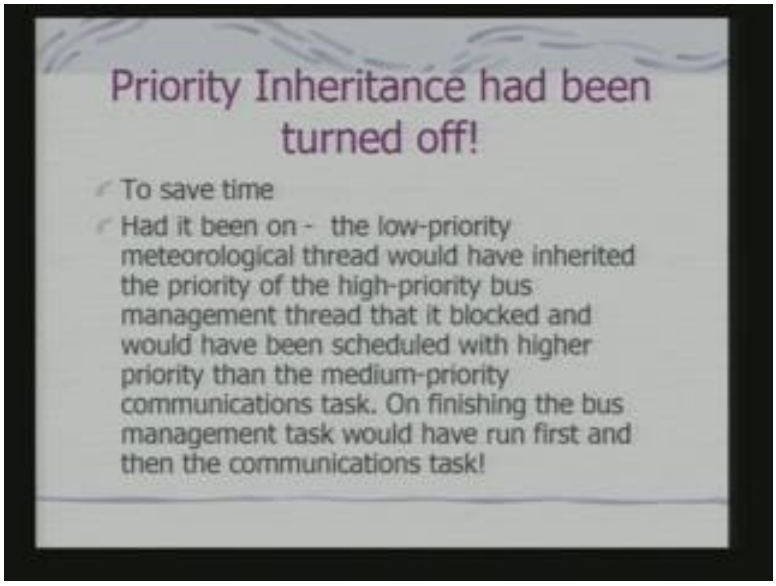
So, what I was trying to illustrate with this example is how the different components in an embedded system are interdependent and they can interact and if you do not take care of all their possible interaction, this can be also be a source of bunk. So, debugging in such a case is extremely difficult.

(Refer Slide Time: 24:07)



You have to do actual execution traces, follow the execution traces to reproduce the scenario and identify the cause and these effects occurred when the rover was actually on mars. And you have to have a complete simulation environment, in the arc station to identify the fault and try to rectify the fault.

(Refer Slide Time: 24:29)

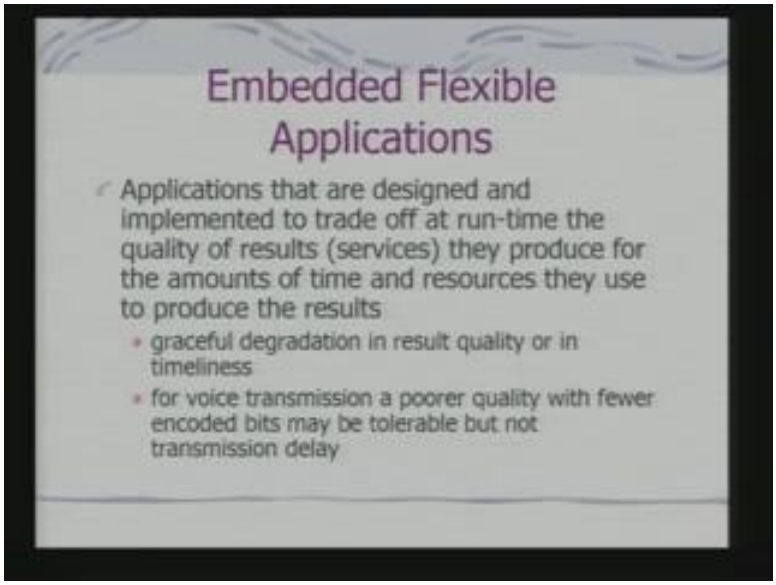


And what was founded was that priority inheritance had been turned off. Since, priority inheritance had been turned off because priority inheritance means what again a kind of a scheduling overhead, in order to save time you have turned off the priority inheritance.

So, had it been on, the low priority meteorological thread would have inherited the priority of the high priority bus management thread that is blocked and would have been scheduled with higher priority than the medium priority communications task.

On finishing the bus management task would have run first and then the communication task. So, the priority inversion period would not have been illuminated because it was illuminated, watchdog timer was finishing its count and that is how the system was getting reset. In fact it is a very well known problem which is cited for illustrating this problems, which gets associated with resource allocation and resource scheduling and the priority assignments depending on the tasks characteristics. Now, you shall look at another kind of applications, which will say flexible applications.

(Refer Slide Time: 25:47)

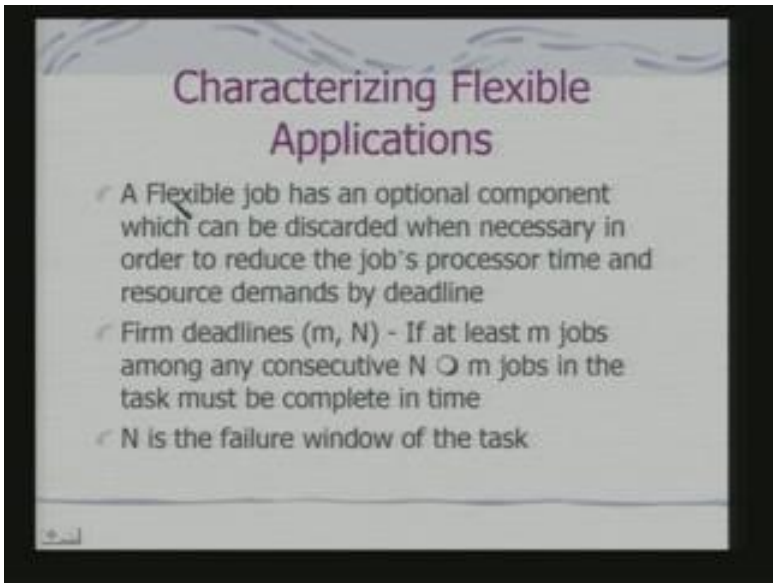


Mainly applications are flexible in the sense that you can actually accommodate what we called graceful degradation in result quality or in timeliness. So, that means what we are talking about is. So, for we have talked about soft deadlines, hard deadlines, soft deadlines I say we can miss sometimes. But, here we are introducing a concept that we may still meet deadline, but the quality of the result may be poor or we may miss the deadline, but still ensure quality of the result.

Now, this kind of flexibility is not there for every, each and every kind of job, for some kind of task this kind of flexibility would be there. So, what do you say applications that are designed and implemented to trade off at run- time the quality of results or services

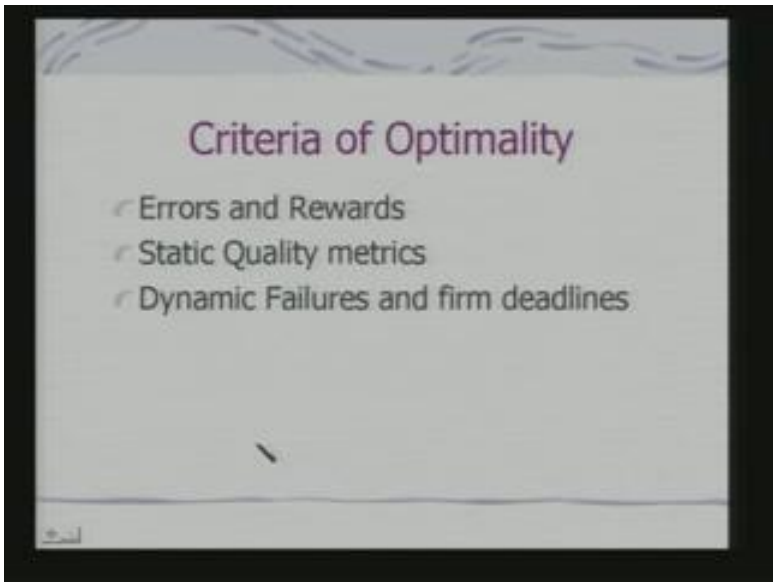
they produce for the amount of time and resources they use to produce the results. So, you say it becomes a kind of a parametric trade off the scheduler can look at the trade off and accordingly scheduled the time, that is processor time as well as other resources.

(Refer Slide Time: 26:57)



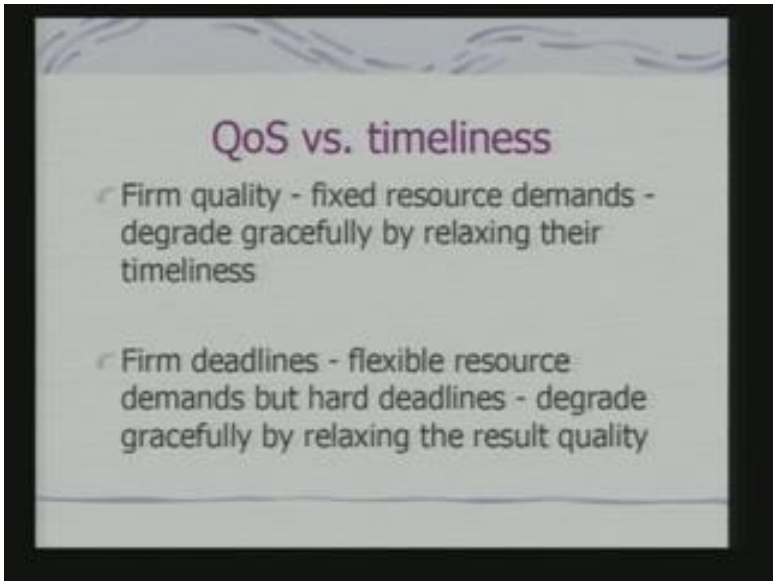
So, how do we characterizing flexible applications, a flexible job typically has an optional component which can be discarded when necessary in order to reduce jobs processing time and resource demands by deadlines. If I talk about firm deadlines m comma N, what is the definition of this, if at least m jobs among any consecutive N into m jobs in the task must be complete in time. So, this is a kind of a deadline, which is getting specify and N is actually the failure window of the task.

(Refer Slide Time: 27:36)



So, what is the criteria of optimality under this conditions, the kind of errors that can commit, the reward corresponding to some of the tasks meeting the deadlines and the quality parameters. And then you have got the static quality metrics. And there could be dynamic failures and requirement to meet this kind of firm deadlines.

(Refer Slide Time: 27:57)



So, we say that there is a freed of between the quality of service and timeliness. So, the how the parameters are related one is called firm quality and firm deadlines. For a firm quality you have a fixed resource demands and degrade gracefully by relaxing their

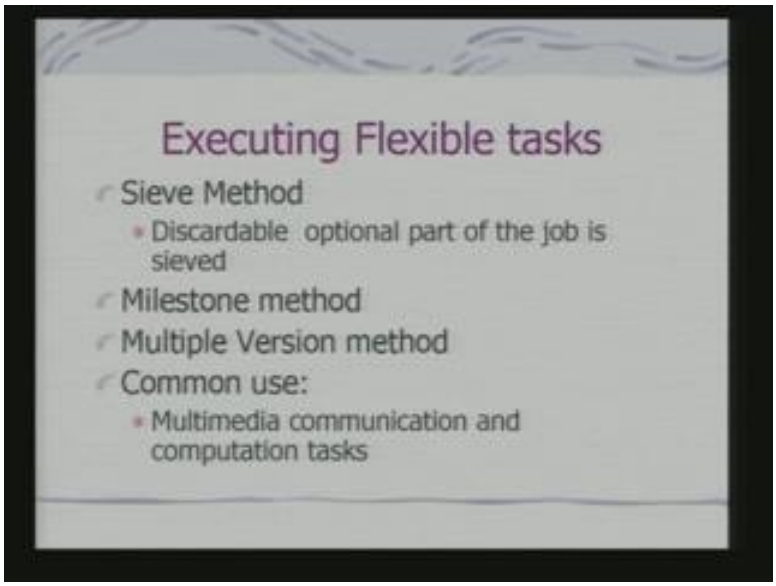
timelines, that means we are still not compromising on the quality, but we are compromising on the deadline.

And on the other side we can have firm deadlines, what is that mean, there is a flexible resource demands but hard deadlines you degrade gracefully by relaxing the result quality. If we take a simple example to understand this, let us consider the job of decompressing of video sequence. Now, when I am doing a decompression, the complexity would be what dependant on the decompression overhead. Now, if the image let us consider a frame is divided in terms of blocks.

And now if I actually do decompression for all the jobs of the frame, I get good quality output. Now, it may so happen that I may not be able to meet the deadline, if I am trying to decompress each and every block. So, I can compromise on the quality, how do I compromise on the quality? I simply copy blocks from the previous frames. I do not decompress each and every block in the image, I simply copy blocks from the previous frame or I just interpolate blocks from the de-complex blocks in the image.

So, here I am meeting the deadline, but compromising on the quality, on the other hand what I can do a video compression task has got soft deadlines. So, I can effort to miss the deadline, by small amount and by missing deadline what I shall do, I shall decompress all the blocks and generate good quality frame. So, these kind of ((Refer Time: 30:16)) is there for the flexible jobs and scheduler, which should take a count this flexibility while scheduling tasks.

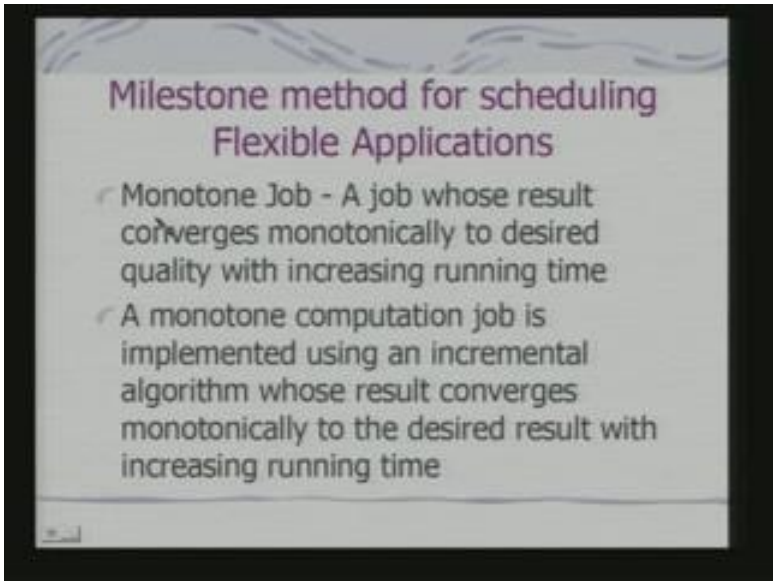
(Refer Slide Time: 30:23)



So, what are the different kind of policies the scheduler can adopt, one is the sieve method, sieve method is the simplest path that is when, a scheduler finds during the schedule ability check that this deadline cannot be met. So, the optional part of the job is discarded, the optional part of the job is discarded, so the computation time decreases and under that condition, it should be possible for the scheduler to generate a feasible schedule.

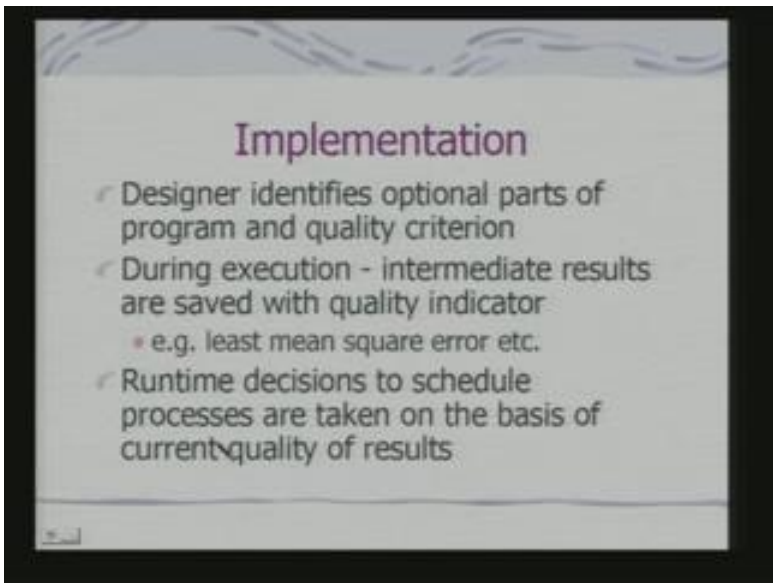
The other methodology is a milestone method, multiple version method, we shall look at them and the very common use of these kind of flexible tasks are related to multimedia communication and computation tasks. And in fact maturity of your multimedia, enabled devices like your cell phone, your vcd player, mp3 players, mp3 players with addict functionality with are coming into market all of them, implement some variant of this kind of flexible scheduling policy for dealing with resources.

(Refer Slide Time: 31:36)



So, what is the milestone method, in a milestone method you deal with what are called monotone jobs, a job whose results converges monotonically to desire quality with increasing running time. So, that means, if I stop at any arbitrary point in time I shall get some result, but I shall not meet the quality. So, a monotone computation job is implemented using an incremental algorithm, whose result converges monotonically to the desired result with increasing running time, that is a basically characteristics of a monotonic job.

(Refer Slide Time: 32:09)

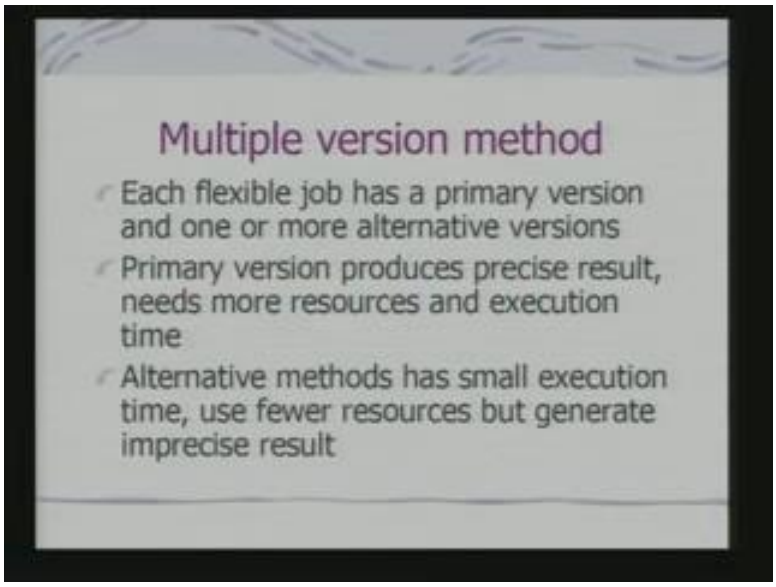


So, how does the monotone for the monotone job this is implemented. So, designer identifies optional parts of program and quality criterion, I already talked about the

quality criterion. During execution intermediate results are saved with quality indicator, it means a least mean square error etc or any other determination for that matter. And runtime decisions to schedule processes are taken on the basis of current quality of results.

That means, scheduler looks at quality of the results and on the basis of that it decides, whether this can be further schedule or not. If, there is some kind of slack will be available it will schedule, the job ones a particular quality is reached. So, what you find, depending on the job beaks, the quality of the result can actually vary then you got a multiple version method.

(Refer Slide Time: 33:00)



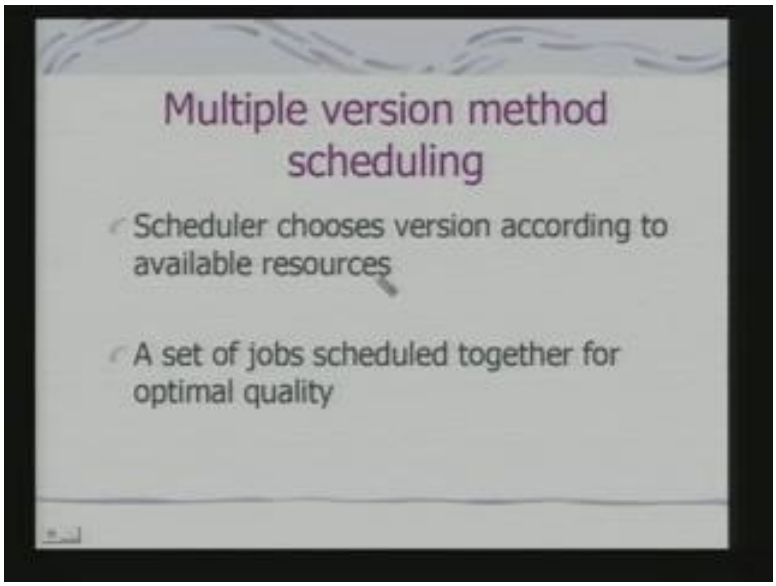
Each flexible job has a primary version and one or more alternative versions. Primary versions produces precise result, needs more resources and execution time. An alternative method has small execution time, use fewer resources, but generate imprecise result. In fact, these multiple version method have coming with the concept of what is called scalable coding for multimedia data.

When you really using any kind of multimedia transmission, there can be a base level coding as well as they can be a enhanced level coding. Base level coding, a simple example could be then if I am using a video phone telephoning application. In the base level I can have a sub sampled frame, I am transmitting at the base level sub sample

frame, when I am transmitting sub sample frame I am requiring less bandwidth, bandwidth is a resource I shall required less processor time to actually show the image.

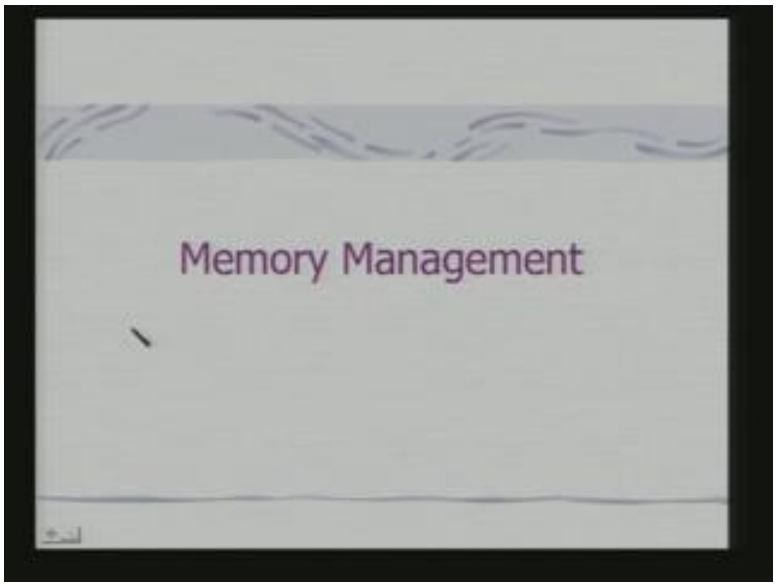
So, that is the base level and you also have an enhanced layer, enhanced layer is the error between the actual frame and the sub sample frame and you transmit that in enhancement layer, so you got multiple versions. So, we have the resource, if you have the resource to accommodate the enhancement layer, you get the enhancement layer as well, if you do not have the resource, you do not use the enhancement layer, you can discard the enhancement layer. So, that is a multiple version method, for dealing with flexible job for resource allocations.

(Refer Slide Time: 34:44)



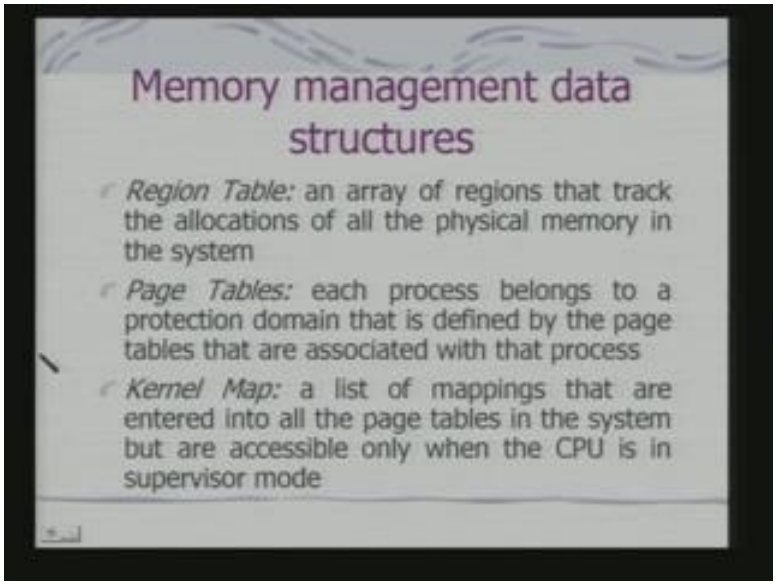
So, scheduler chooses version, according to available resources, so whether it could accept for our example, accept the base layer or the enhancement layer. And set of jobs are scheduled together for the optimal quality.

(Refer Slide Time: 35:00)



So, this more are less, finishes our dealing with resources in a generic sense. So, how resources are managed, we have looked at when there is a conflicting demand for the resource, how the priority have to be dealt with. We are looked at, when there are job characteristics such that the resource usage can be made flexible, how that flexibility can be exploited for scheduling the resources. Now, we shall look at features of two basic resource, one is memory other is generic IO device.

(Refer Slide Time: 35:38)



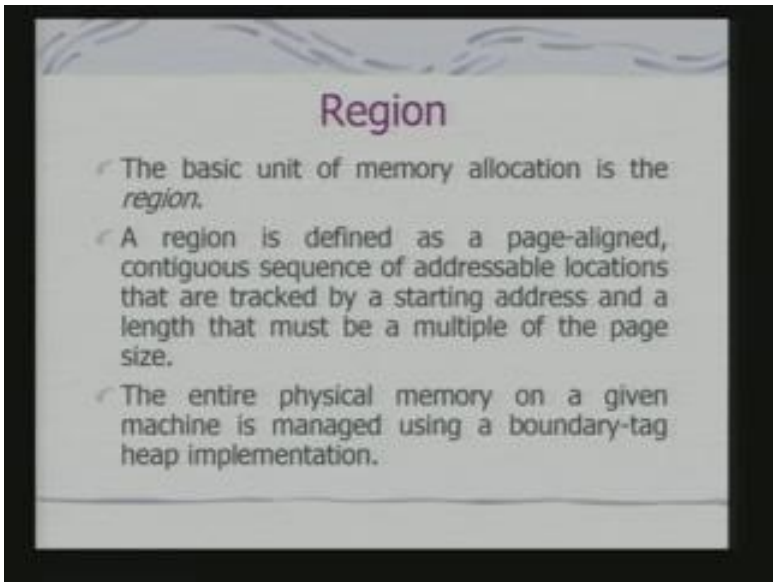
The memory management, typically in an embedded system involves on an embedded OS involves managing the right you take as a data structures. Because primary job is that of managing memory allocation between concurrent tasks and in fact, when we looked at the memory management unit in the context of Rm we have already seen that one job of MMU is to provide protection to the memory areas allocated to different tasks. The data structure which are use for this purpose, we in a generic sense we say region table, page tables and kernel map.

Region table is an array of regions that track the allocations of all the physical memory in the system. Page table is each table belongs to a protection domain that is defined by the page tables that are associated with that process. In fact, each processes got associated with the page table and that can be associated with the production domain; that means, the production features with the pages allocated to the process, when there is a concept of what is called a kernel map.

Kernel map is a list of mappings that are entered in to all the page tables in the system, but are accessible only when the CPU is in supervisor mode, what does that mean see kernel itself the ways will map on to a set of pages, these pages are meet part of the page table of each and every process. Why should you do that, because processes can need to be, ((Refer Time: 37:20)) called to the OS.

So, in that case you need not do a exclusive context switch, what you can actually do is use this pages, but the protection comes in place because these code which are in this pages are executed, only in supervisory mode, a region is a basic unit of memory allocation.

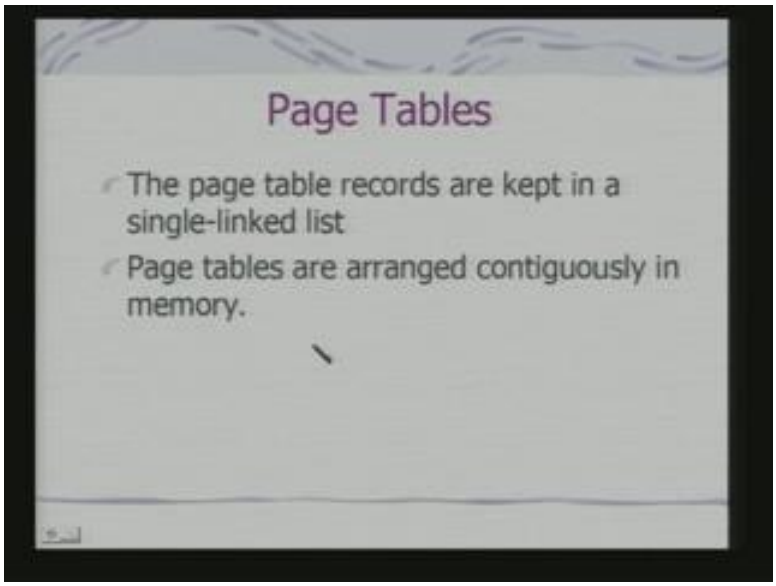
(Refer Slide Time: 37:43)



And the region is typically defined as the page aligned contiguous sequence of addressable locations that are tracked by a starting address and a length and that must be multiple of page size. So, actually you will find this region, how it comes in a, if you remember in the context of arm we talked about sections and these sections in a generic way we are referring to as regions. The entire physical memory on a given machine is managed using a boundary tag heap we have talked about heap in the context of processes.

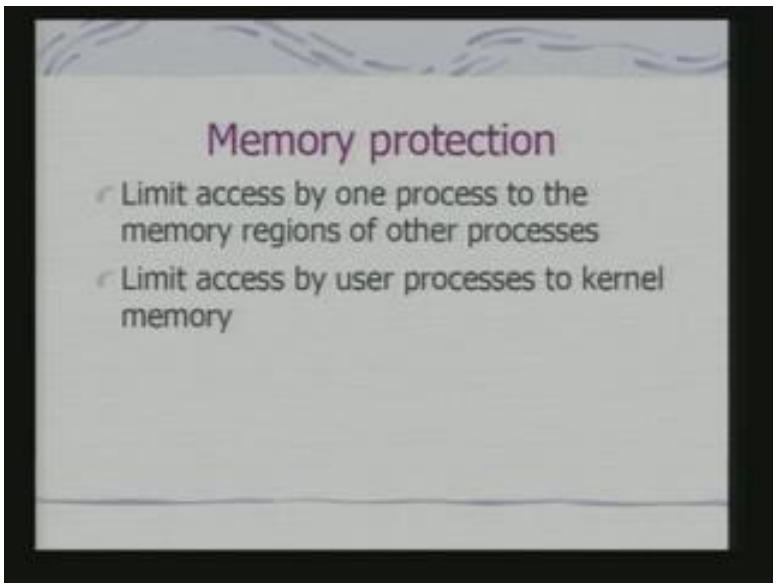
Now, OS has to manage the complete memory area, for allocating memory to the different processes. So, how does it manage such memory, it manages that memory again using a heap like a data structure, but heap now stores basically the region data and the region information. And the region typically in a paged memorizes system would be hump size would be what a multiple of the page.

(Refer Slide Time: 38:44)



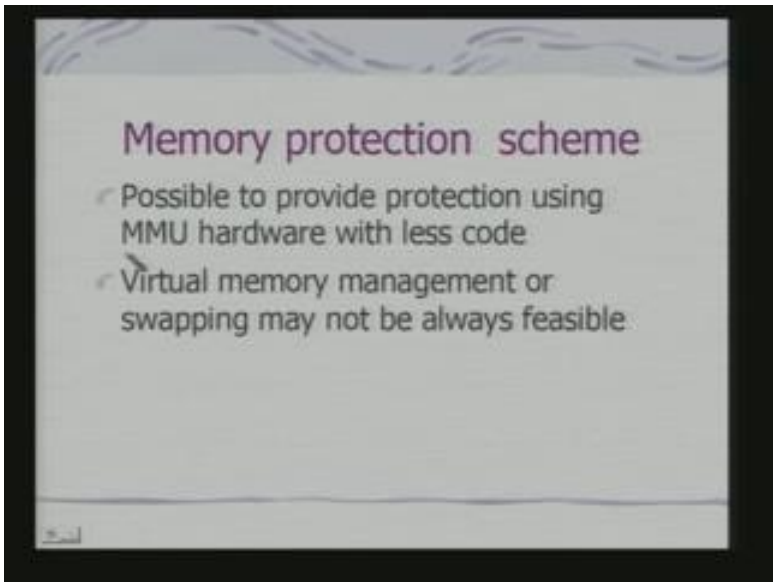
Page tables are records are kept in a single linked list and page tables are arranged contiguously in memory corresponding to a process.

(Refer Slide Time: 38:53)



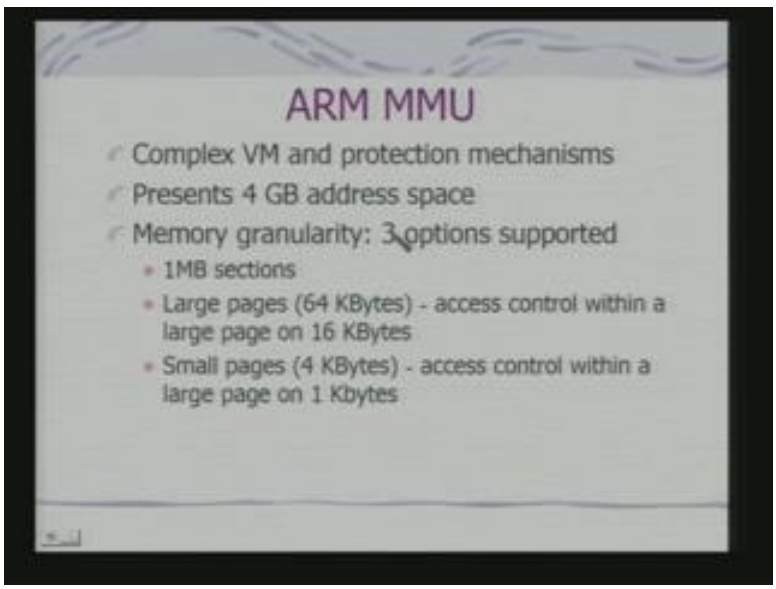
So, using all this data structure fundamentally and the most importantly in the embedded system, which service is provided it is a security service. So, that in one process can not actually destroy or corrupt memory of some other process. So, two issues is there therefore, limit access by one process to the memory regions of other processes. And limit access by user processes to kernel memory.

(Refer Slide Time: 39:25)



And the memory protection scheme today, if it is at all implemented in embedded system in majority of the cases and exploits the MMU hardware. And virtual memory management may not always involve swapping. Now, in this case because in many embedded systems, you really do not have secondary storage like this. The swapping, I had already discussed earlier, the swapping can between flash and the RAM and if at all there is a disk, between the disk and the RAM. But in many cases, this swapping part of the virtual memory management system is not implemented, what is primarily made use of is a protection mechanism.

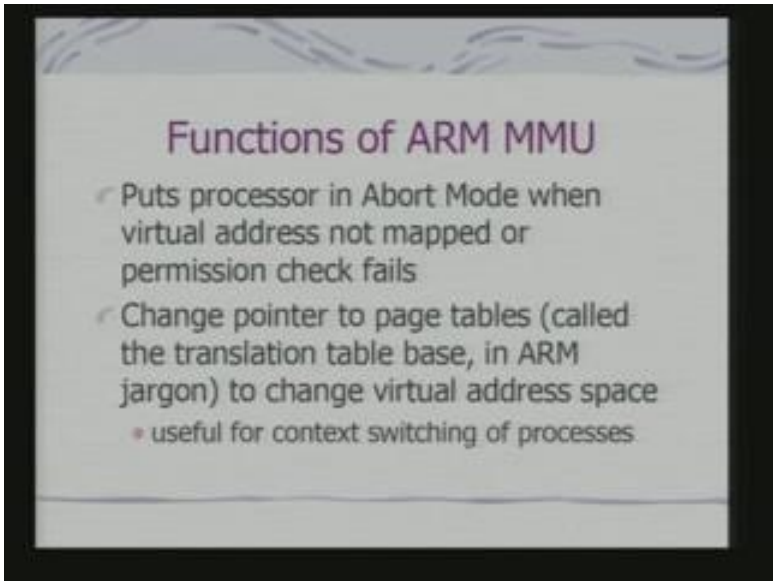
(Refer Slide Time: 40:07)



See, we go back to ARM and recap what we are talked about the ARM MMU actually provides for this. So, OS actually does what, has the code for running the different functions that MMU offers. If you remember that it presents ARM MMU can man manage 4 GB address space, depending upon the size of the address bus and memory granularity 3 options 1 MB sections large pages, small pages.

So, these kind of pages, collection of pages or sections are being referred to as regions. And region can consist of a number of pages and OS keep track of the regions and the pages, for the purpose of allocation to different processes. And what does MMU do for protection, it defines what are called various kinds of domains and using domains actually, this protection policy is implemented.

(Refer Slide Time: 41:02)

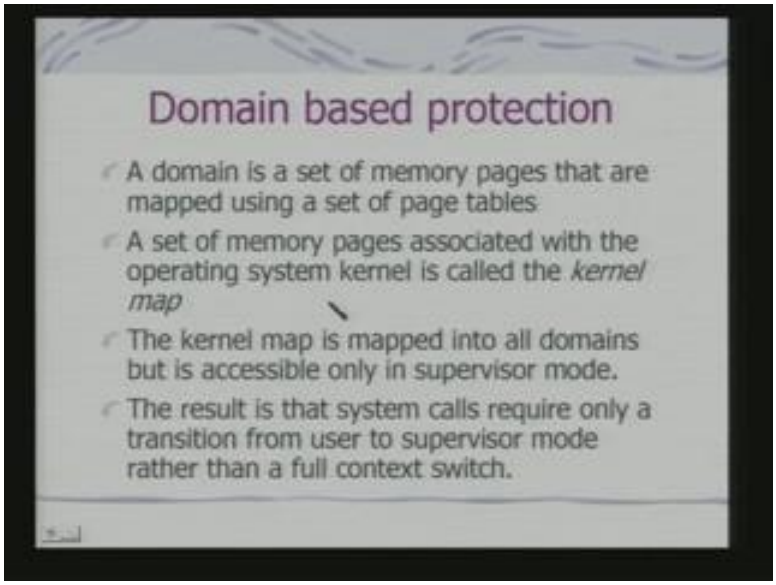


Apart from that, these other tasks which are done through MMU is when there is a virtual address is involves virtual memory and swapping between a secondary storage and the primary storage like RAM. So, a processor put a processor, what we say in abort mode, when virtual address is not mapped to permission to check fails, so that is an abort mode of arm. So, OS has to provide the code for the abort mode for actually page transfer from secondary to primary memory.

So, these apart of the code will run in the abort mode. Other thing is for normal task management change pointer to page tables called translation table base, because these

happens when processes switch takes place because it will switch from one page table base to another page table base and page table bases are managed by the MMU.

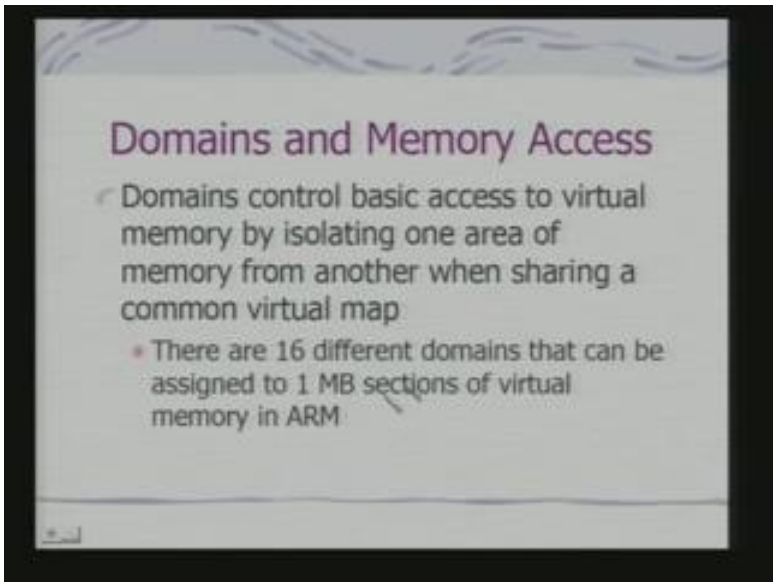
(Refer Slide Time: 42:00)



Now, we come to the domain base production because MMU is also provides the domain base production. In fact domain is a set of memory pages that are mapped using a set of page tables. A set of memory pages associated with the operating system kernel is called kernel map. The kernel map is mapped in to all domains, but is accessible only in supervisor mode, result is that system calls require only transition from user to supervisor mode rather than of full context switch, this point is we have already touched about.

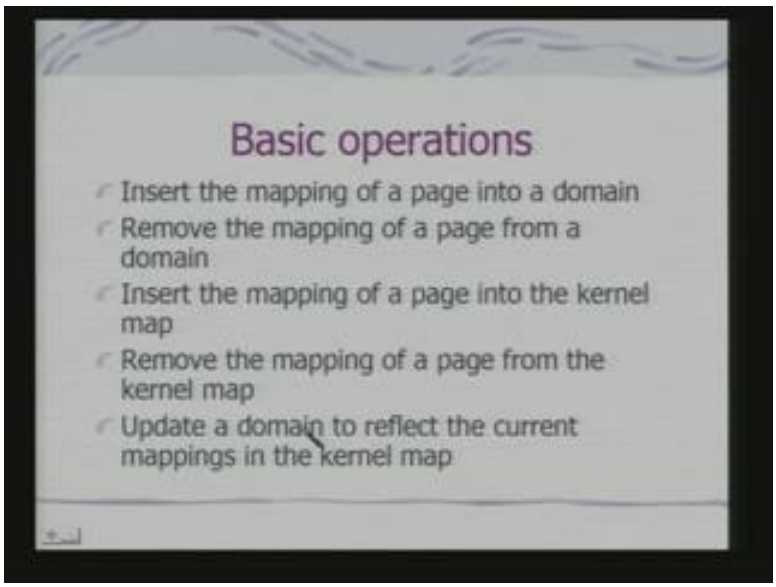
But what is the basic concept of domain, domain is each process is basically a collection of pages, which can include pages belonging to other processes like kernel of the OS. Now, therefore that set of pages, are associated with the domain base protection. So, domain defines the protection features, the access rights. So, that gets defined associated with the domain. And MMU checks that whether your ((Refer Time: 43:03)) the way there accessing pages which are belonging to a domain, whether the process is violating any of the domain related access control conditions.

(Refer Slide Time 43:18)



So, domains control basic access to virtual memory by isolating one area of memory from another when sharing a common virtual map. In fact, there are 16 different domains that can be assigned to 1 MB sections of virtual memory in ARM. So, these are the different kinds of protection scenarios, which can be implemented by the OS on the virtual memory.

(Refer Slide Time: 43:41)

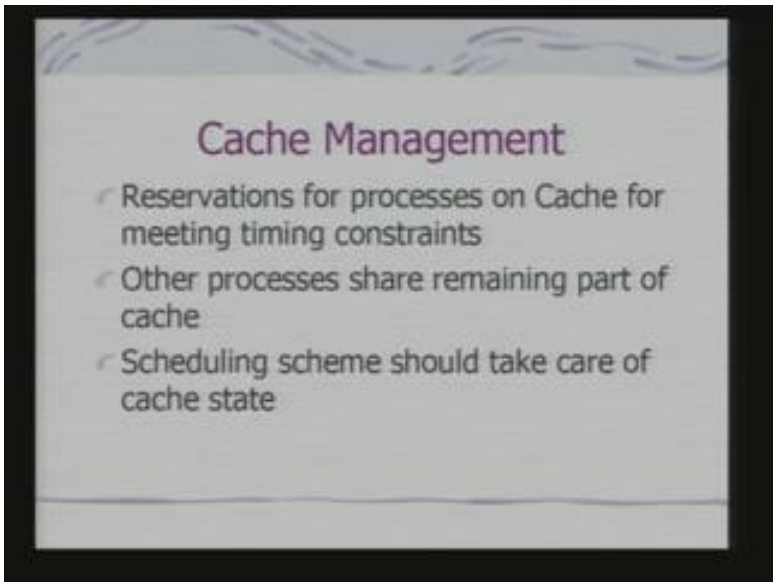


So, basic operations, for the domain is inside the mapping of the page in to a domain. Remove the mapping of a page from a domain. Insert the mapping of the page into the

kernel map. Remove the mapping of a page from the kernel map update a domain to reflect the current mapping in the kernel map. So, that means, there can be pages now belonging to different processes. So, you can consider even if there is a shared memory, just like information bus we are talking about for the rover.

That shared memory would be again a page, which may belong to multiple domains for the purpose of an exclusive access. But, when the access takes place there is really know context switch, but access may takes place depending on where and how the shared memory is ping managed. If it is part of the kernel, the access can takes place via the kernel map.

(Refer Slide Time: 44:34)

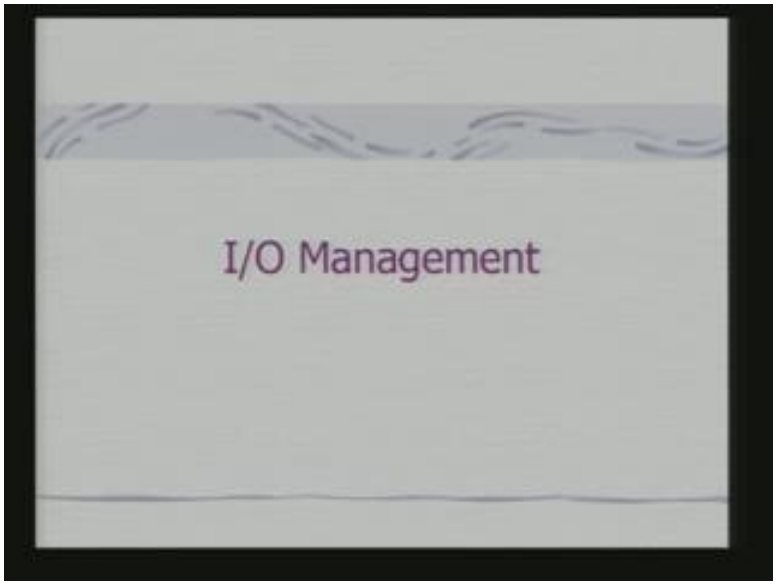


Cache management is another important aspect of the OS. One of the issues of the cache management is that when we are looked at the question of computation time and you can understand the computation time of a process can actually vary depending on whether the process is in a cache or not. Now, we may like to optimize on this computation time. So, we may decide to have a clean partitioning of the cache memory and so I can have locking facility in the cache memory.

So, lock facility is there, lock facility can be in the hardware, but it is managed by the OS. And in the design time you may decide that these task is the critical task and hence it should locked in the cache, it may be a kernel task may be the scheduler, which should be locked permanently in the cache, so that it is never soaped out. The other resource

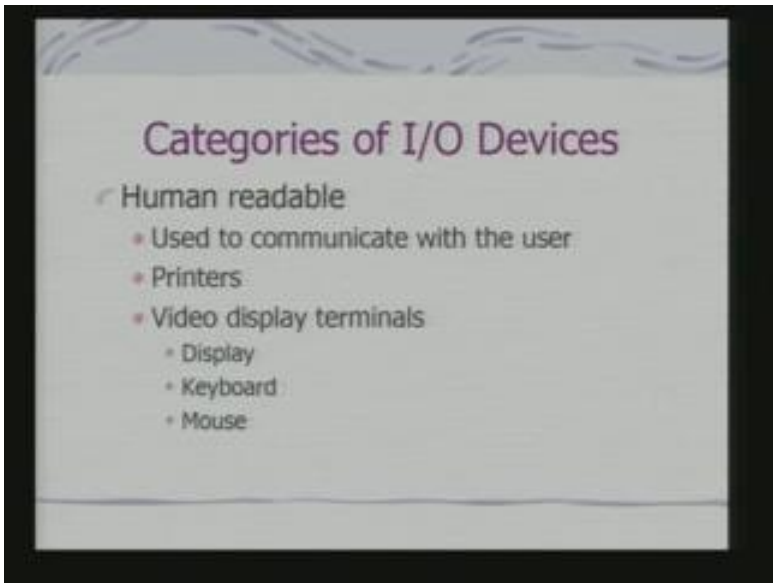
which is to be managed because, since our embedded system is situated in an external environment which deals with variety of IO devices.

(Refer Slide Time: 45:45)



So, managing IO becomes a critical issue.

(Refer Slide Time: 45:50)

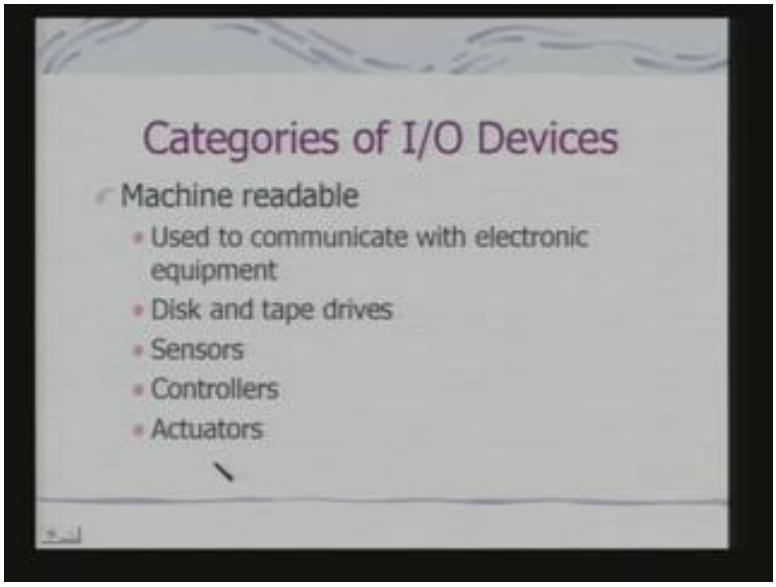


In fact, IO devices can be categorized into various categories, because the management also varies on the basis of their need. So, human readable examples are printers, video display terminals, which has got a display, keyboard, mouse all of these are IO devices. And these kind of devices we encounter not only on general purpose computers, but if

you are looking at an embedded systems like sony game station, video games systems there are also these kind of devices at to be managed.

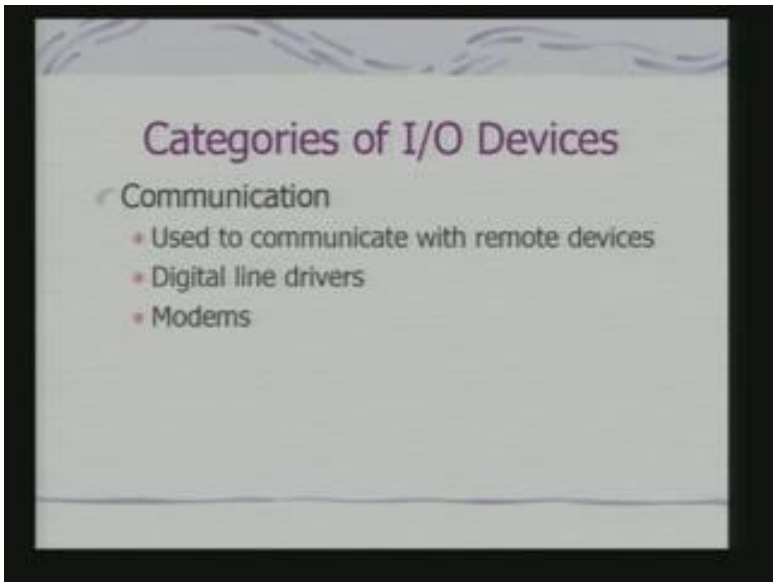
Printers are typically you will find in point of sale terminals, there you have got small printers for doing calculations and providing the resist, ATMs got printers for providing the resist. So, these it is not that these kinds of devices are only found with general purpose computers, but also an embedded systems. So, we need to know how do you deal with these kind of devices.

(Refer Slide Time: 46:47)



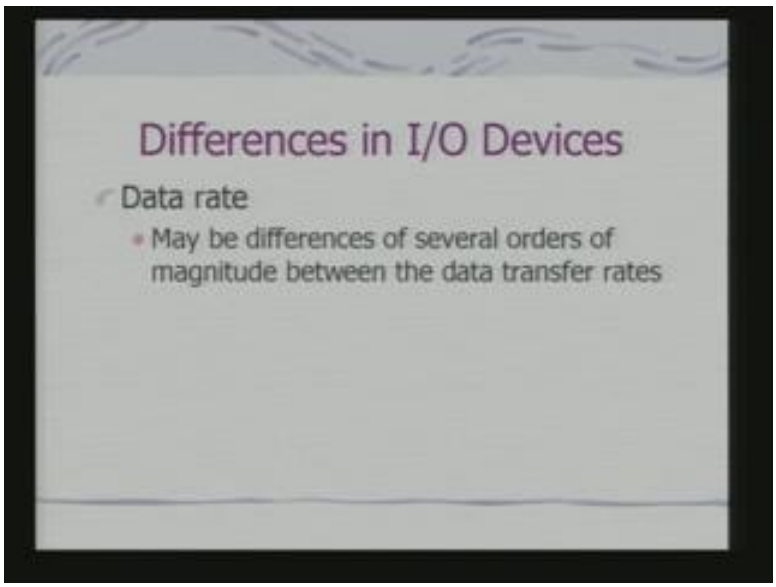
There are machine readable, sensors, controllers, actuators they are more common with the embedded systems. Disk and tape drives are obviously, that common, but disks are also there in a variety of forms with since the size is reducing and the storage is increasing.

(Refer Slide Time: 47:08)



And then communication devices, which are primarily to communicate with variety of things, so what we are looking at, we are looking at three categories of devices. Why this categories are coming in because of their nature of the functions to operate. Then, the nature of the data that they handle would be different and accordingly the interfaces has to be designed.

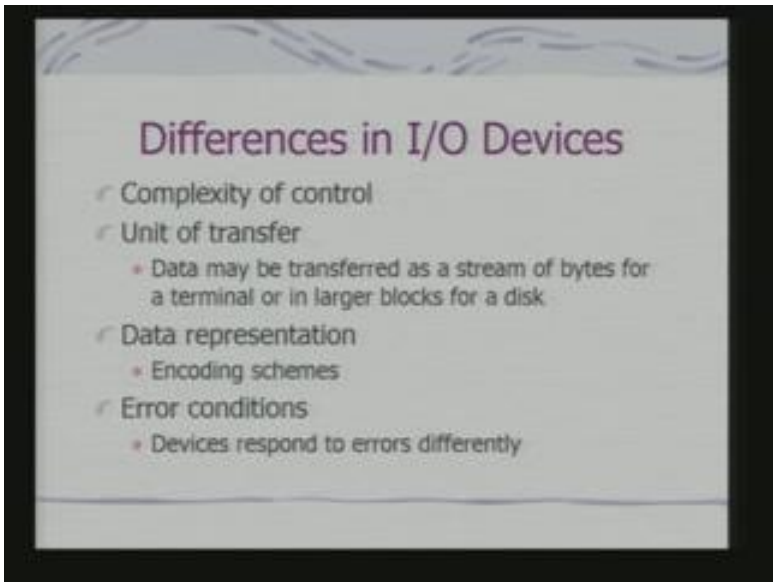
(Refer Slide Time: 47:30)



So, the difference is really come in several orders of magnitude between what you called data transfer rates. Why this data transfer rate is important because this data transfer rate

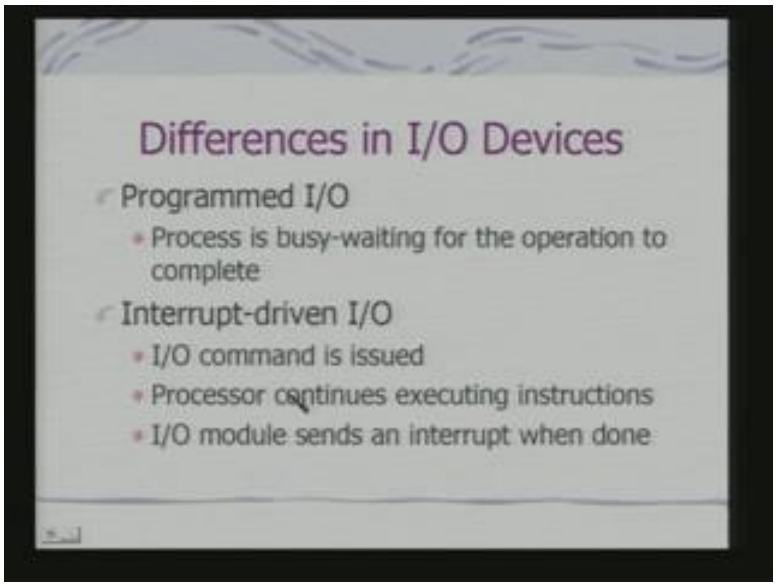
is related to interrupts, that this device can actually generate. And you have the processes which are also running is it that will you permit interrupts to interrupt execution of a critical task for each and every data transfer corresponding to a high speed device, that is the decision that the OS has to take while designing the software.

(Refer Slide Time: 48:16)



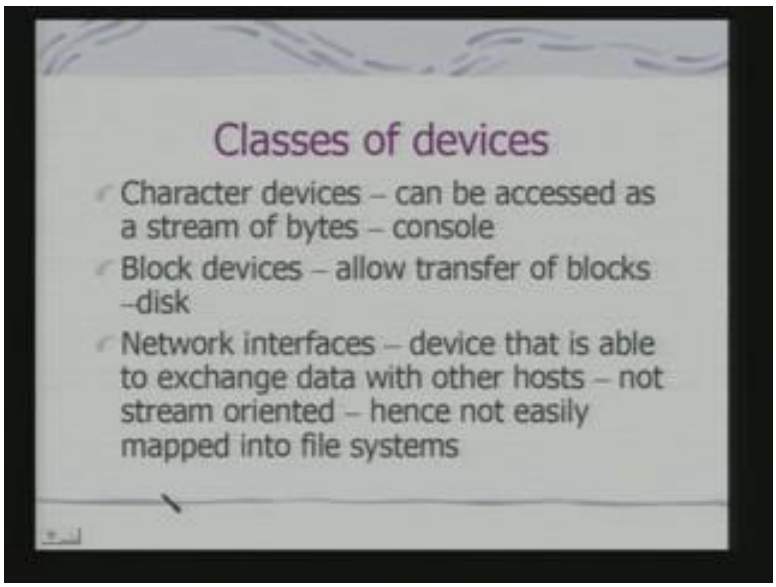
So, there would be a complexity of control related to the device and the key issues are unit of transfer, the data may be transferred as the stream of bytes, may be it may have as a blocks like it is transfer from a disk, then you have got encoding schemes as well as error conditions, because devices respond to errors differently. And on an embedded system you have to have provision for taking care of this exceptional condition.

(Refer Slide Time: 48:43)



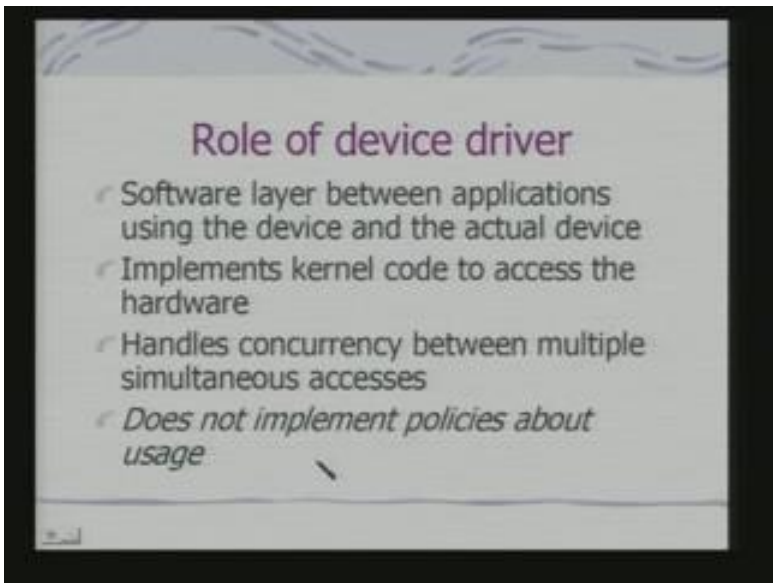
And typically you have programmed IO and interrupt driven ((Refer Time: 48:49)) because you interrupt device interfaces therefore, has to cater to these two requirements as well as if there is a DMA, how does a DMA gets initiated. DMA gets initiated by enabling the DMA controller, who enables a DMA controller a kernel level task. So, OS has to know how to initiate also the DMA, it has to these interrupts driven IOs also has to associated with this ISRs, this ISRs also your part of your OS and programmed IO this is also part of your OS.

(Refer Slide Time: 49:22)



So, broadly if you look at this kind of functionalities, the devices we can put in these three categories. Character devices can be accessed as a stream of bytes, it can be a typically console, block devices allow transfer of blocks typically disk. Network interfaces typically communication devices, device that able to exchange data with other hosts not stream oriented, hence not easily mapped in what you say in a file system.

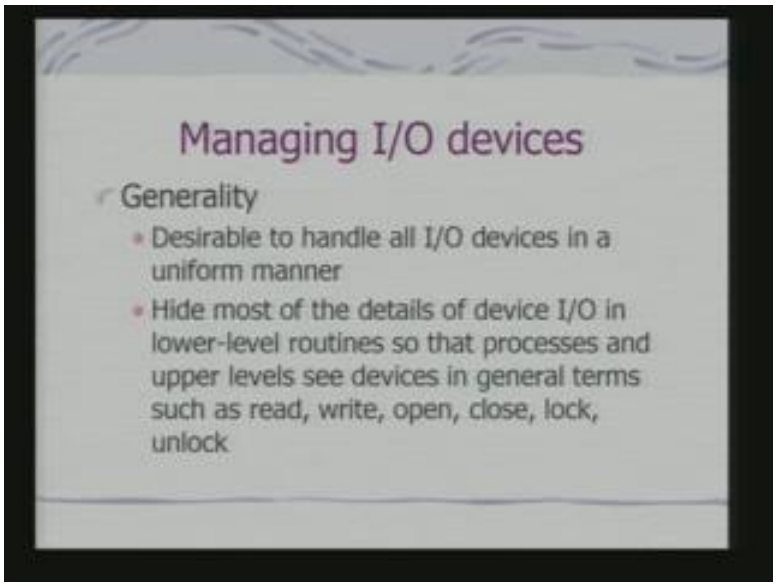
(Refer Slide Time: 49:51)



Now, you have got the device driver, so all these things are managed basically having device drivers. Device drivers provide what, the software layer between applications using the device and the actual device and we are actually seeing that each device has got its own typical characteristics. So, OS has to provide for this kind of device drivers, catering to needs of each and every device. So, device drivers implements kernel code to access the hardware. It handles concurrency between multiple simultaneous accesses.

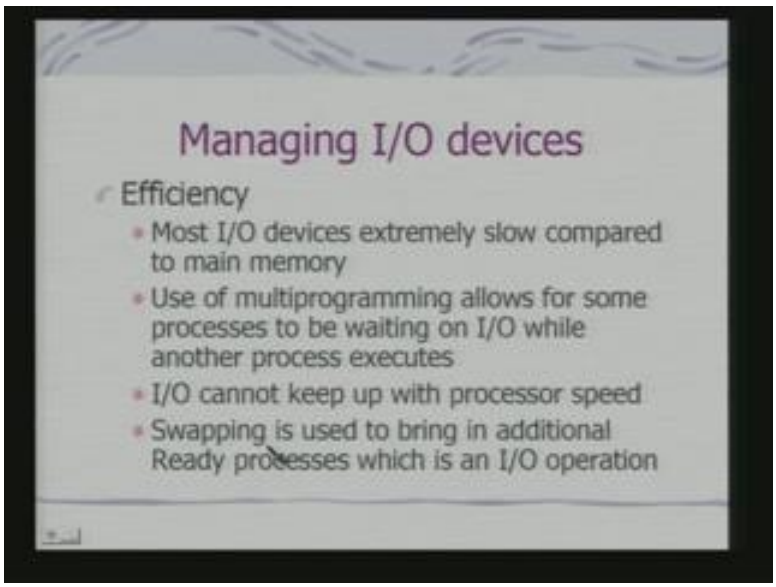
But it does not implement policies about usage, because this policies comes from where, from the resource scheduling. So, resource we are already talked about resource scheduling policies, in terms of set priority ceiling, priority inheritance that not part of the device driver, device driver just provides an interface. So, basically this device driver should be written in which way, they should have some kind of a generic capability.

(Refer Slide Time: 50:57)



Generic device drivers, desirable to handle generalities that because we need to handle all IO devices, we would like to handle IO devices in an uniform manner, so in that case you hide most of the details of the devices IO in lower level routines. So, those applications that use read, write, open, close, lock, unlock this kind of top level routines, which are not really concern with details of the device.

(Refer Slide Time: 51:23)

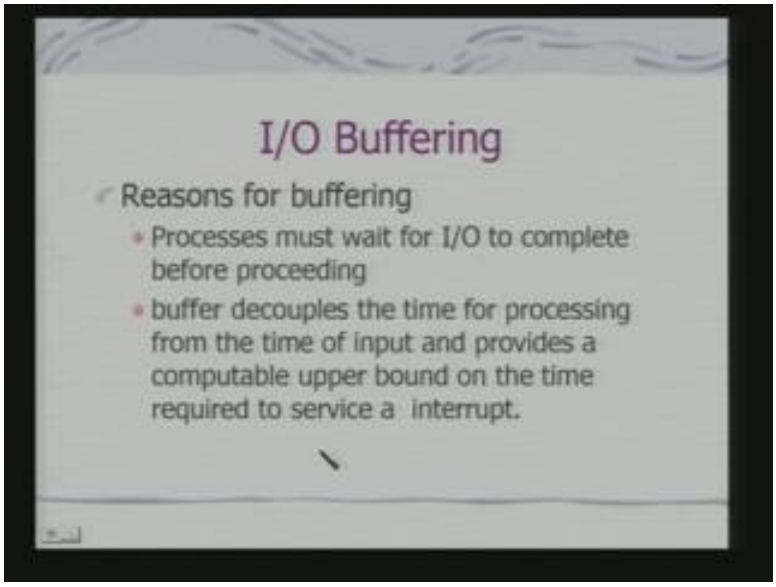


Efficiency is also an issue and this is a key issue why, because most IO devices are extremely slow compared to main memory, when if you have got a first device that slow

and use of multiprogramming allows for some processes to be waiting on IO while another process executes. So, blocking can happen when you not being able to meet the requirement of the data of a process from a device, in fact typically all device access requires give rise to blocking.

And swapping is used to bring in additional ready processes, which is in IO operations if you see swapping, if you are doing it some flash are from a disk can be actually considered as an IO operations.

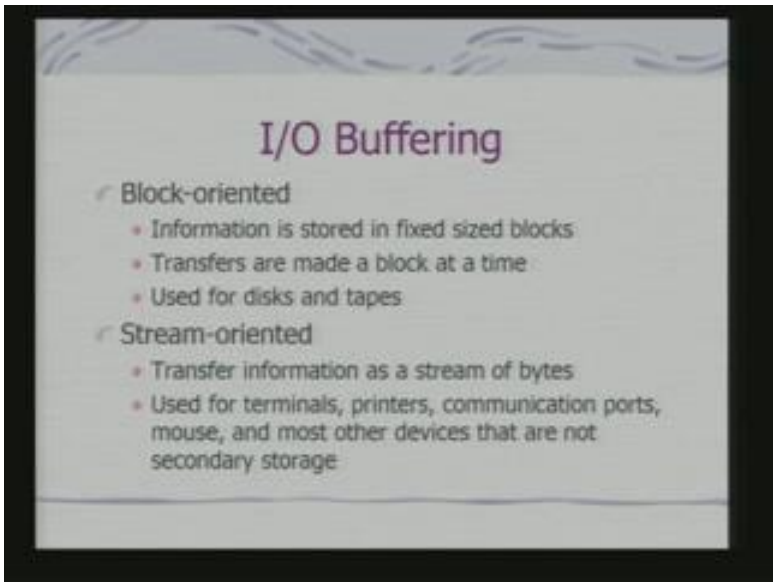
(Refer Slide Time: 52:11)



So, what you do is for any basic policy that OS adopts for interfacing with these devices is buffering and these buffers are actually managed to your device drivers. So, processors must wait for IO to complete before proceeding, but you can use a buffer what you say. The buffer decouples the time for processing from the time of input and provides a computable upper bound on the time required to service an interrupt.

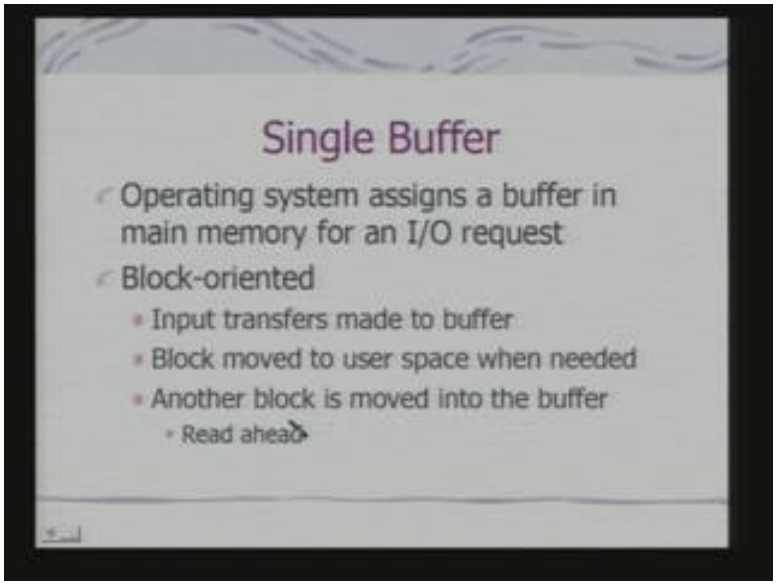
Because you actually have a buffer, if I consider a communication device, a communication device is getting the data and writing on to the buffer. And when a buffer is full a communication device may be generating and interrupt. So, the device is not generating an interrupt at each and every point in time, when it is receiving in a data and there could be variety of offering schemes because these buffers are again resources which are manage by the OS. This buffering can be block oriented or stream oriented.

(Refer Slide Time: 53:18)



Block oriented because you have got devices, which are block oriented device. So, in this case in formations are store in terms of blocks and transfers takes place in a block. Stream oriented is why need information is a transfer as a stream of byte for terminals, printers, communication ports etc.

(Refer Slide Time: 53:40)



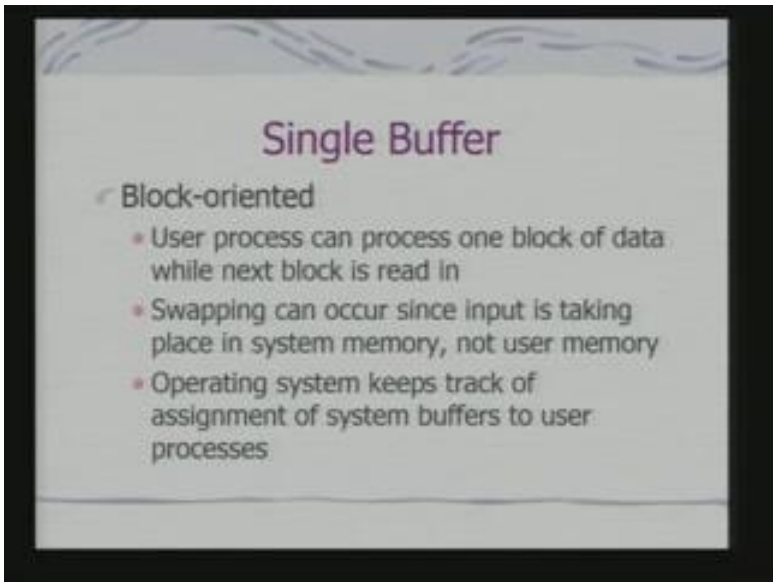
Now, this buffering can also coming in variety of ways, the simplest is single buffer. Operating system assigns a buffer in main memory for an IO request. So, in a block oriented device what happens, the input transfers are made to the buffer and blocked is

move to users, space when needed. And another block is moved into the buffer. So, you are doing a read ahead.

Now, how this movement takes place, this movement needs if the buffer is a page, you just alter a page table entry in the page table of a process. If the processes asked for a data from a disk or may be from a flash, you actually get the data on to a buffer that data transfer, now you can consider I can use simply a DMA to do this transfer. When a DMA is doing this job, processor is not at all involved, but what processor has done, processor has allocated that buffer.

When the DMA is complete what happens DMA control generates an interrupt, when a DMA controls generates an interrupt, the interrupt service to do can actually do what depending on which processes asked for the IO can make corresponding entry in the page table. Once it makes entry in the page table, the domain base production comes in place. So, this data is also gets protected from unauthorized access from different processes.

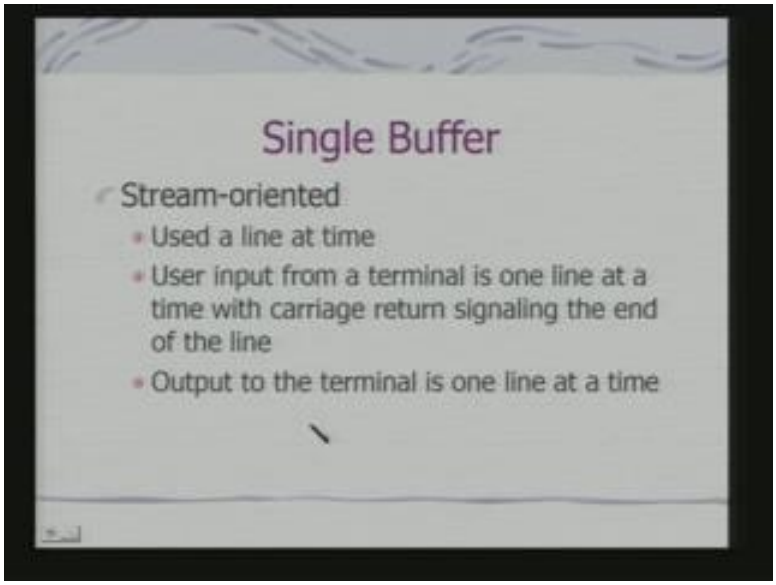
(Refer Slide Time: 55:11)



This block oriented what you say the user can process one block of data, while next block is read in because the movement one block is allocated a block is given another block the OS can make schedule. Swapping can occur since input is talking place in system memory and not user memory. And operating system keeps track of assignment

of system buffers to user processes that is how exactly the paged table modification is done.

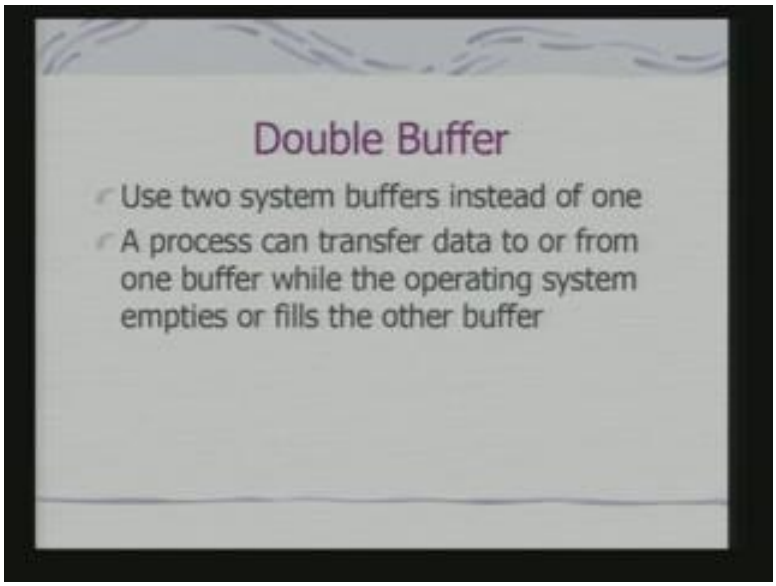
(Refer Slide Time: 55:36)



Now, when it is stream oriented, different from block oriented what happens, you used a line at a time. So, user input from a terminal is one line at a time with carriage returns signaling the end of the line. So, output to the terminal is also one line at a time. So, for each character you can see the keyboard interface, it is not that is word each character actually the data is ready, you allocate the buffer of a size and you do the reading, when there is a new line.

So, that specifically the typical feature of a stream base interface and if you remember that we have said that, we have classified what basically the devices as block based stream based. So, when I have the stream based buffer. So, the buffer management is stream based and who manages the buffer, the device driver corresponding to that device. In fact, the interrupt service to determine comes in that becomes part of your device driver.

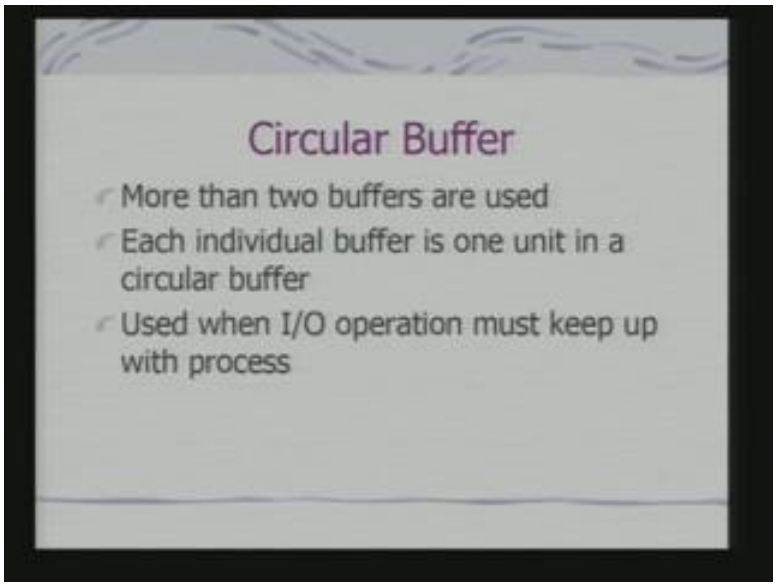
(Refer Slide Time: 56:39)



In many cases you use double buffer, use two system buffers instead of one. So, process can transfer data to or from one buffer, while the operating system empties are fills the other buffer. Because it can be therefore, buffer actually becomes times share because what happens, while one buffer is being filled up for a normal case, the process which has requested the IO is blocked, till the buffer is full. And the buffer gets transferred, it will cannot access that if get a double buffer than both operations can takes place simultaneously.

So, in fact providing the next buffer is almost similar to that of the double buffer, but double buffering means that at any point in time two buffers are typically allocated. In a single buffering, one buffer is allocated and once there is a mapping of that one buffer to the user process, the next buffer gets allocated.

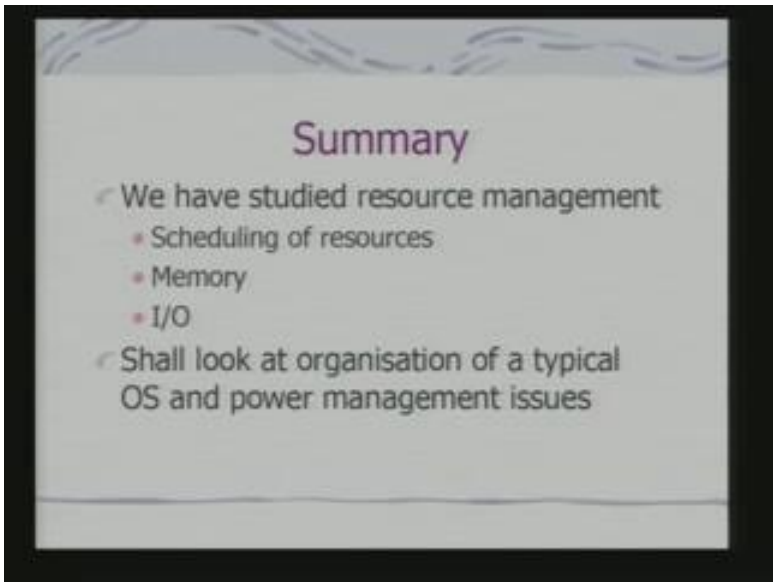
(Refer Slide Time: 57:40)



This can be done also through circular buffer. More than two buffers are used in case of a circular buffer and each individual buffer is one unit in a circular buffer. So, it is a continuous stream of data. When use a circular buffer, effectively you are giving an impression of an infinite storage because you are entering data from one in and going through just like a circular queue. So, that kind of a continues stream of data management becomes easier.

So, if you are talking about a kind of a speech interface for a cellular phone the buffer allocation for the receiver would be a circular buffer, because you really do not have an idea of what will be the length of data and it will be a continues process. So, this buffering is the key issue in terms of generic aspects of IO management and other aspects becomes device specific.

(Refer Slide Time: 58:39)



So, we have studied basically the resource management, primarily scheduling of resources, memory as well as that of IO. In the next class, we shall look at organization of a typical OS, which is targeted for embedded appliances, as well as we shall study the issue of power management because so for we have considered memory, we have considered IO, we have considered processors for the purpose of allocation. But, power is also a very important resource in an embedded system, which is not true that a general purposes system. So, you shall look at that point in the next class.