Information Security-3

Prof. V Kamakoti
Department of Computer science and Engineering
Indian Institute of Technology Madras

Basics of Unix and Network Administration
Operating Systems Introduction
Mod02 Lecture 09

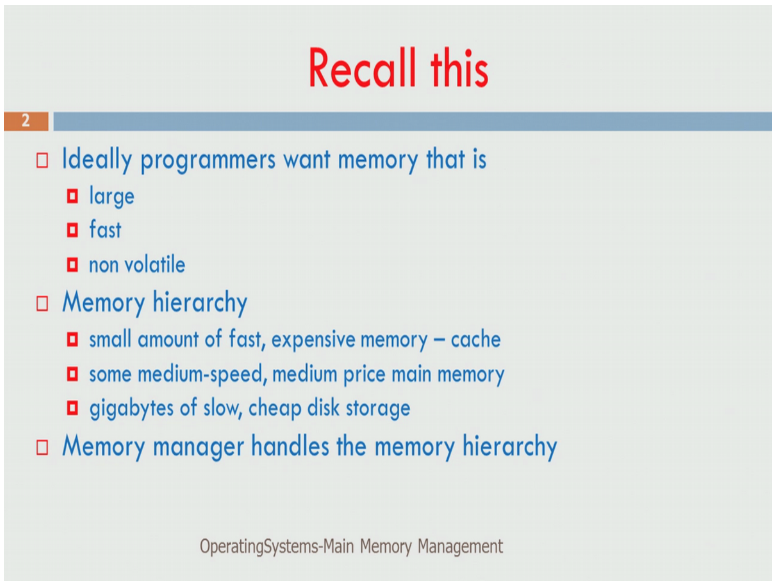
Module 9: Memory Management (1)

So welcome to module 9 and we will be talking about memory management when we look at a execution of an instruction, we actually see that there are five stages in the execution of an instruction, one is to fetch the instruction, increment the program counter. The second step is to decode the instruction. The third step will is to fetch the relevant data for that instruction, fourth is to execute the instruction, fifth is to throwback the result. Out of these three stages, fetching the instruction, according to Nyman (())(0:50)model of computation, the program that is to be executed should reside in the memory. So I have to fetch the instruction from memory, I have to also fetch the relevant data from I might also need to fetch the relevant data from memory and then I am at also want to store the data back into memory.

So there are 3 important, steps in these 5 steps, which basically looks at memory or which needs memory. So managing memory is a very very important aspect of you know operating systems overall management put (())(1:28) you and from a security point of view also memory management is extremely crucial. So in the information security 2 course, we had actually seen about inter and intra process protection against process A trying to access process Bs data or code or stack and vice versa and even within process A process A trying to exit the limit of memory allocated to it. So there were lot of architectural support that was available, which can basically detect any type of malfunctioning of the process in terms specifically from the terms of protecting the memory that is allocated to it and ensuring that it is used correctly. So we had seen quite a bit of that in information security 2 course.

Now what there we had talked about what can an architecture do towards protection of the memory that a process is basically allocated by the operating system. So it was more from an architectural perspective the security and protection can be enforced on memory is allocated to processes. In this module, we will see the operating systems perspective. So in (the) this module 9 and module 10 is memory management 1 and memory management 2 we will look at different functionalities of the operating system with respect to managing memory in terms of allocation of these memories to processes and retrieve in a back. So that would be the subject content and as I mentioned earlier 3 of the 6 steps involved we will basically touch memory and so memory management is extremely crucial.

(Refer Slide Time: 3:19)



Recall this

2

- Ideally programmers want memory that is
  - large
  - fast
  - non volatile
- Memory hierarchy
  - small amount of fast, expensive memory – cache
  - some medium-speed, medium price main memory
  - gigabytes of slow, cheap disk storage
- Memory manager handles the memory hierarchy

OperatingSystems-Main Memory Management

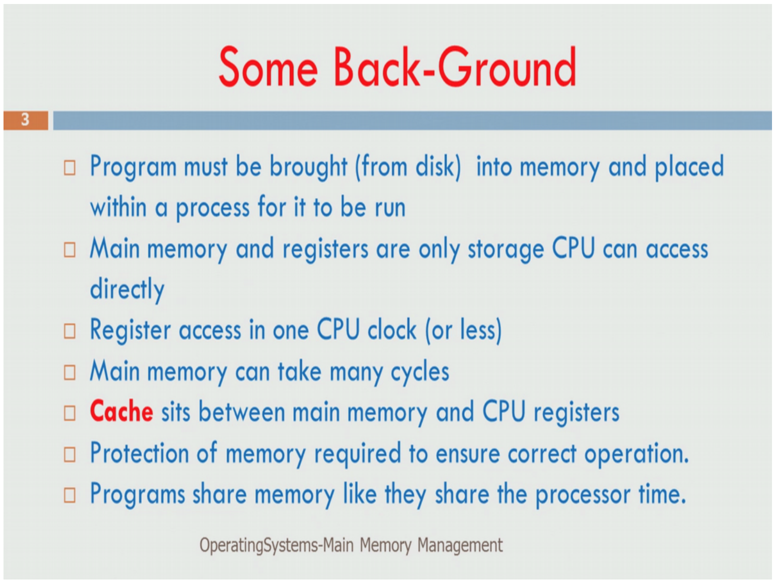
Now what is the programmer want? Since, as per the Von-Numan (())(3:23) model, the program that is going to be executed should reside in memory and the CPU only talks to memory. They need programmer wants to write large programs, A wants it to be executed fast and if there is of A wants the program to be stored for multiple executions. So 3 requirements from memory is that it should be large, it should be fast and it should also be non-volatile that is when there is a powers cut the program that we have written should be again readable at a later point of time.

Now when we look at these three parameters. These are 3 orthogonal parameters if I want very large memory then it cannot be fast if I want non-volatile memory then it cannot be you know it cannot be both fast and large. So if I want a non-volatile memory I could have it on fast, but it cannot be large or it can be large, but it cannot be fast. So all these 3 parameters essentially or 3 orthogonal parameters trying to improve 1 will (())(4:26) at the other 2 and so striking the balance and managing them **is** is going to be the biggest task of the operating system.

So to manage these 3 parameters the entire system is built on what we call as a memory hierarchy. So the CPU is a chip inside the chip there is some amount of memory, which is very fast but it is not non-volatile. It is a volatile memory and it cannot be too large. It is a small but fast but volatile memory, right and this is the cache, cache or the registers that you have inside the chip.

Now after that there is a large memory, but it will be slower than what we had seen the cache and it will be volatile and that is the main memory. After this there is a large memory which will be much slower, but non-volatile and that is the disc and tapes. So there is a memory hierarchy, there is a non-volatile large slow memory which is the disc, there is a volatile, little faster but and little less larger memory, which is the RAM and then there is a volatile very fast, but very small memory which is the cache and the registers and so as we move away from the CPU if we are within the CPU the same chip then your size is going to be small and your, but your access is going to be fast and as I start moving away from the CPU to the RAM then to the disc the your size is going to becomes larger, but your speed is going to reduce and of course the volatility is achieved with a disc storage. So manager apart of the operating system ha so handle this memory hierarchy in the best way.

(Refer Slide Time: 6:27)



Some Back-Ground

3

- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are only storage CPU can access directly
- Register access in one CPU clock (or less)
- Main memory can take many cycles
- **Cache** sits between main memory and CPU registers
- Protection of memory required to ensure correct operation.
- Programs share memory like they share the processor time.

OperatingSystems-Main Memory Management

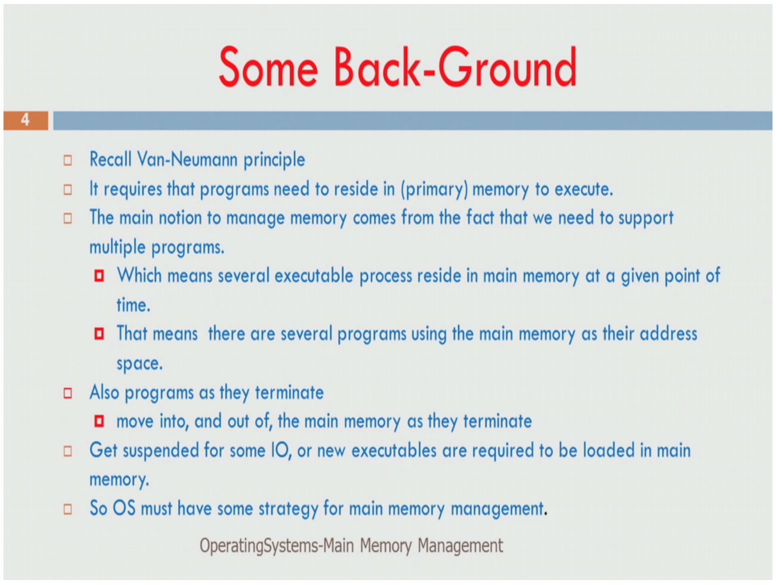
So this is the von Numan (())(6:23) stuff, the program must be brought from the disc into the memory and placed within a process for it to be run. So the when the program is brought into the disc, from the disc into the memory then it actually becomes a process as I had mentioned earlier process is a program in execution and so the program starts executing then it actually becomes a process.

Now the main memory and registers or the only storage, which the CPU can access directly. So the CPU cannot access the disc, right. So the register access can happen within one CPU clock or even less than that. It is a very fast. So the main memory can actually take many cycles while the cache is the actually seating between these registers and the main memory. So it takes a couple of cycles. Now one of the important thing is that what a so cache is a replica of the main memory. So there should be both needs to be protected in the sense that when the CPU wants to access from main memory, some copy of some of the relevant data and instructions would be inside that the cache. So the first CPU will go and check the cache if it has the relevant data or the relevant instruction if it is there then it will access from the cache access means it can read or write into that cache if it is not there then only it goes to the main memory.

So there is always a possibility that the content of the main memory and the content of the cache are different, for example, let us say address 1000, 1000 in the main memory may store 50, but the same 1000 would have been fetched into the cache. So and the CPU would try to write into 1000 and it would have made it say 48, right. So there is a different between what is stored in the cache and what is stored in the equivalent address in the main memory and this needs to be addressed. So this is one correctness that we need to ensure and this actually called as cache coherency issue and so other thing is that there are two processes in execution when process A is executed the corresponding caches of that our field when process B is executing it may be allocated to some other memory locations when it starts executing in the CPU Process A cache need not be completely invalidated, because process A will come back to execute at a later stage and that whatever is there in the cache is necessary for it to go and execute there in a fast manner.

So when process B is executing, process A addresses would have been cached the content would have been cached and so process B should not be in a position to access those cache entries. So these are all some of the issues that we need to keep in mind when we want to ensure protection between two processes and that is also very important from an information security point of view. So these are some of the background things, so I assume that all of you know what a cache is what a In-memory is and how cache works. These are all very basic things. Now from that as a background I am posing certain problems in the area of information security.

(Refer Slide Time: 9:52)



Some Back-Ground

4

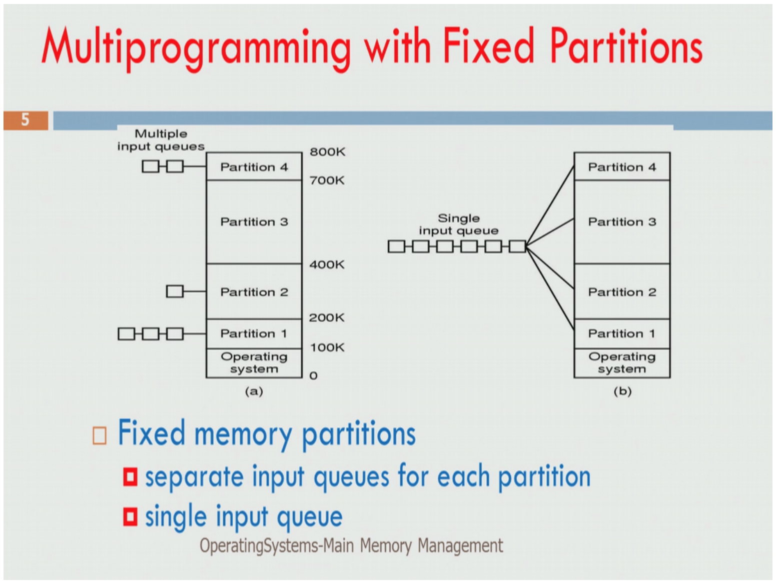
- Recall Van-Neumann principle
- It requires that programs need to reside in (primary) memory to execute.
- The main notion to manage memory comes from the fact that we need to support multiple programs.
  - Which means several executable process reside in main memory at a given point of time.
  - That means there are several programs using the main memory as their address space.
- Also programs as they terminate
  - move into, and out of, the main memory as they terminate
- Get suspended for some IO, or new executables are required to be loaded in main memory.
- So OS must have some strategy for main memory management.

OperatingSystems-Main Memory Management

The other important aspect, so when a process into existence the operating system gives it some memory. So the process needs some memory for storing its instructions storing its data and also the stack. So at least 3 sungs (())(10:07) of memory are given by the operating system to a process that comes into execution. Now when the process completes its execution, now the operating system has to reclaim this. So essentially you should reclaim the data and stack and the memory allocated so that it could be reuse for some other process and this is also very very important aspect of memory management and when I reclaim and allocated to some other process, some data that is written by the old process should not be seen by the new process. So should be go and completely null it, if I if every memory that is released if it is going to be nulled and then given to some then there can be some issues in the sense that there can be performance issues and so these are all some of the decisions, the operating system needs to take from an information security perspective. So there is a process A executing it releases the memory.

Now this memory is allocated to process B, now should I nullify this or just keep it off. So these are some of if I nullify then there are some performance issues if I do not nullify then there are some security issues. So these are some of the balances that we need to strike (()) (11:17). So this is also some very important aspect of information security from the memory management perspective of operating system.

(Refer Slide Time: 11:26)



Multiprogramming with Fixed Partitions

- Fixed memory partitions
  - separate input queues for each partition
  - single input queue

OperatingSystems-Main Memory Management

Now this is how things work when multiprogram the system. What is a multiprogram system more than one process are ready to execute at any point of time, then for example, mail server. So there are many people who are logged in many processes will be in execution. Suppose I say there is an only one CPU some amount of CPU time will be given by the operating system for user 1 each process will be pulled out and then user 2 will be given then user 3 then user 4. This is basically scheduling a round robin scheduling essentially happens.

So there will be different programs that are basically trying to get executed at the same point of time and for each of these programs, there will be some memory that is allocated, right. The way operating system allocates this memory, there are multiple ways, so one of the thing is this fix at memory partitions, for example there could be say 3 input queues as you see on your left hand side. So one of the queue will be allocated partition 4, another queue will be allocated partition 2, another queue will be allocated partition 1. So this type of a partitioning even from an information security perspective we could say that we could have different ways of reclaim policies for partition 1 as a extremely secure programs can execute only in partition 1.

So now you see that in partition 1, there are 3 programs that are allocated memory in partition 1 if it all they need memory they should take it from partition 1 and whenever the memory is released that will be completely nulled, right so that so extremely secure programs can be seating in partition 1. Similarly, there could be something in partition 2, right. So partition 2 may be completely insecure programs like meaning I do not want that are that level of security. Now partition 4 can be you know again it can be either secure or say not secure. So this is what we call as multiple partition based allocation of memory and a program can a process can be allocated to one of these queues and we can be allocated.

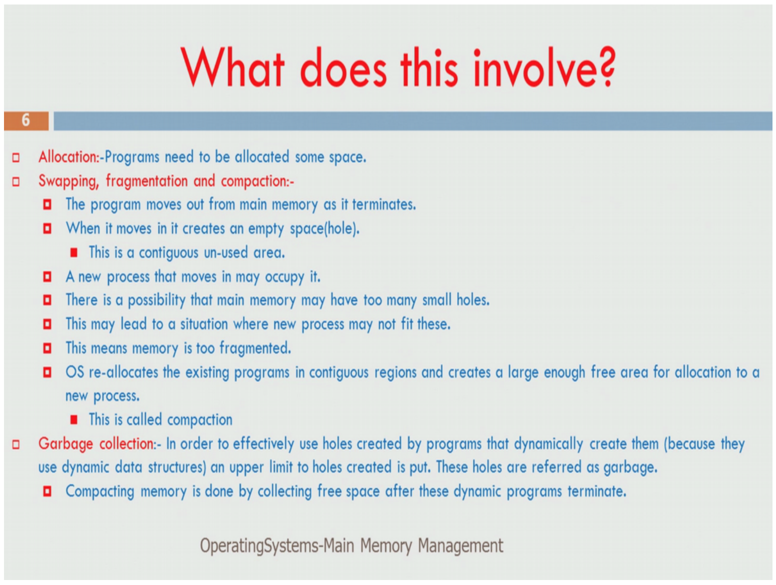
The other way is to have a single input queue and we can allocate it to any of these partitions, right. So again the drawbacks between the what you see on the left of your screen and what you see on the right of your screen is that, in the left hand side, if suppose there are no processes for partition 4, right that path the entire memory of partition 4 will be remaining vacant (())(13:55) while there is big queue for partition 1 then there will be not enough memory to serve it. So the essentially these the longer queue in partition 1, the waiting time for the process will increase, right.

So and at the same time there will be so much memory in partition 4 that is vacant (())(14:13), but we cannot do this shifting there, because we are now said this is fix at the queue 2, which you can go and fix it (())(14:20), on the right hand side we see, all the partitions are available for all the processes except some part which will be occupied by the operating system as you see or the bottom most part of the memory.

Now that time of you know memory wastage will not happen. So the so every memory will be used, but at the same time I could not have difference reclaim policy or scheduling policies for the different partitions in the memory. So these two are 2 different concepts, so even in modern systems where we are trying to enforce security, there we could designate different modules of the memory as one as very trusted memory or process can only trusted process can access those memory and we can say an untrusted interested memory, we can create now we call it as partition we can also call it as zones. So I could have different memory zones one zone is (())(15:11) extremely protected on only certain process can do and the other zone can be extremely free to use.

in the information security 2 course, we had actually seen memory can be allocated to one of the 4 privilege levels, so I could have privilege 0, which is extremely trusted the privilege 1, privilege 2, privilege 3 and in the so I could have different privilege levels and I can also ensure that a privilege 3process cannot touch privilege 0 etcetera. So we had use segmentation and or paging to basically achieve this in the context of an x-axis (())(15:46) architecture. This was demonstrated through labs in the information security 2 course.

(Refer Slide Time: 15:52)



What does this involve?

6

- Allocation:-Programs need to be allocated some space.
- Swapping, fragmentation and compaction:-
  - The program moves out from main memory as it terminates.
  - When it moves in it creates an empty space(hole).
    - This is a contiguous un-used area.
  - A new process that moves in may occupy it.
  - There is a possibility that main memory may have too many small holes.
  - This may lead to a situation where new process may not fit these.
  - This means memory is too fragmented.
  - OS re-allocates the existing programs in contiguous regions and creates a large enough free area for allocation to a new process.
    - This is called compaction
- Garbage collection:- In order to effectively use holes created by programs that dynamically create them (because they use dynamic data structures) an upper limit to holes created is put. These holes are referred as garbage.
  - Compacting memory is done by collecting free space after these dynamic programs terminate.

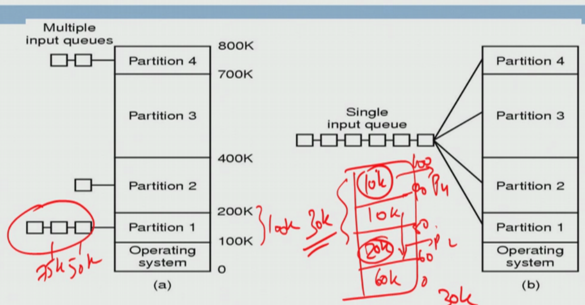
OperatingSystems-Main Memory Management

So once we do this type of allocation, so there are different things that are very very important here, the 3 important things are one is swapping, another is fragmentation and another is compaction. So what is swapping?

(Refer Slide Time: 16:06)

Multiprogramming with Fixed Partitions

5



- Fixed memory partitions
  - separate input queues for each partition
  - single input queue

OperatingSystems-Main Memory Management

So let us go back to the previous slide, so let us take partition 1 here I have 3 processes that are pending here. Now let us say each process needs say, one needs 50 k another needs 75k, right. 50k is allotted. Now when 75k wants to execute there not be enough memory, because there is only totally 100k here, so what we need to do is when the 75k process has to execute means 50k has to be moved into swaps space, so that is what we call it as swapping.

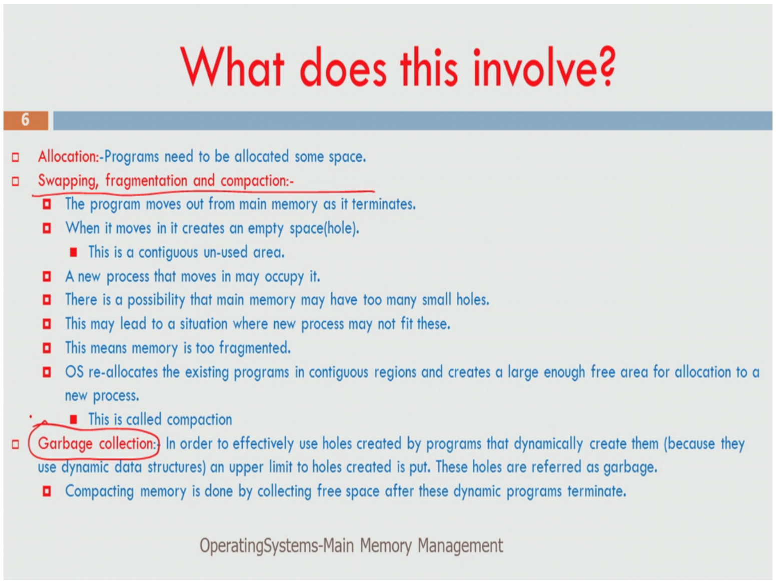
So swapping is very very important, right. Allocation first which process goes to which partition is 1 allocation, after doing this allocation I may have to swap, because there may not be enough memory available to execute, so I cannot say that process 2 will wait till process 1 should complete then only process 2 can come it that is also wrong, because when the process 2 can internally wait if process 1 takes long long time, right. So I need to do a sort of round robin scheduling here, by process 2 also should start executing. So when process 1 is executing it consumes 50k of memory. Now when process 1 is removed, now process 2 starts executing that will prepare (())(17:17) 75k of memory. Now what will happen for this 75k, I have only 100k, 50k is already occupied by process 1, so I have to remove process 1 I have to swap it and then bring it back. So swapping is very very important here.

The third important one is allocation, the second is the swapping, the third important thing is what we call as fragmentation. So let us say, so there is say 100k of memory. Now process 1 comes it occupies say, 60k process 2 needs 20k it occupies, process 3 needs 10k, it occupies there, right. Now process 4, so this is 0, 60, 80, 90, process 4 also needs 10 it also occupies this 10k, so this is 100, okay.

Now let us say that process 4 finishes that means this goes off and process 2 also finishes this finish. Now what we need is what and the total amount of memory now available is 30k, right because process 2 and process 4 finish. Now there is process 5, which is asking for 25k, so I have 20k here left and there is 10k here left, now there is a new process, which is asking for 25k? Now I cannot allocate this 25k process here, because I do not have contiguous memory which is 25k, right. So what I need to do when I shift this 10k ground then I create 30k here on the top to which I can give you 25k, I can give 25k to the process (())(19:06) 5, right.

Now that means what has happened here? When these two process 2 and process 4 left the system essentially, it has created some holes or what we call as fragments and, because of this fragmentation, though I have 30k of memory and a new process only wants 25k of memory I am not in a position to give that 25k, because I need to compact it and then only I can give. So this notion of fragmentation and compaction is very very important from memory management perspective.

(Refer Slide Time: 19:43)

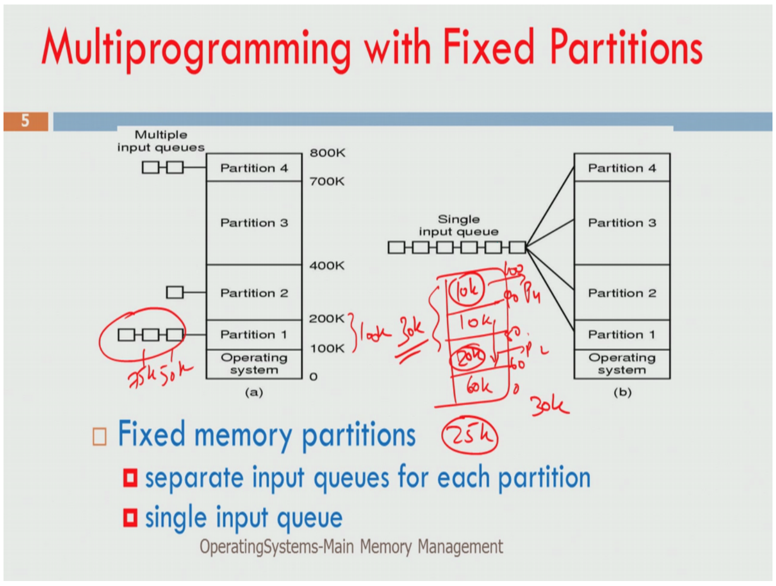


What does this involve?

6

- Allocation:-Programs need to be allocated some space.
- Swapping, fragmentation and compaction:-
  - The program moves out from main memory as it terminates.
  - When it moves in it creates an empty space(hole).
    - This is a contiguous un-used area.
  - A new process that moves in may occupy it.
  - There is a possibility that main memory may have too many small holes.
  - This may lead to a situation where new process may not fit these.
  - This means memory is too fragmented.
  - OS re-allocates the existing programs in contiguous regions and creates a large enough free area for allocation to a new process.
    - This is called compaction
- Garbage collection:- In order to effectively use holes created by programs that dynamically create them (because they use dynamic data structures) an upper limit to holes created is put. These holes are referred as garbage.
  - Compacting memory is done by collecting free space after these dynamic programs terminate.

OperatingSystems-Main Memory Management



Multiprogramming with Fixed Partitions

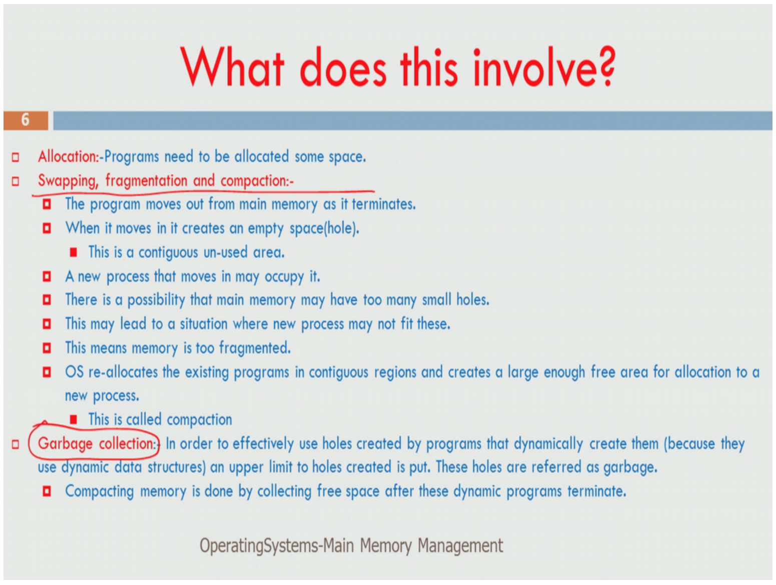
5

- Fixed memory partitions 25k
  - separate input queues for each partition
  - single input queue

OperatingSystems-Main Memory Management

So the 3 important things that we see here basically swapping, fragmentation and compaction and what we did when we compacted essentially we collected we call this garbage collection, right something that has got over, so as we see in the previous slide, this 10k and 20k which was occupied by process 2 and process 4, process 2 completed process 4 also completed. Now those are all useless things so that is why we call it as garbage. So we collected it as one junk which is that 30k and then we started working on this.

(Refer Slide Time: 20:15)



What does this involve?

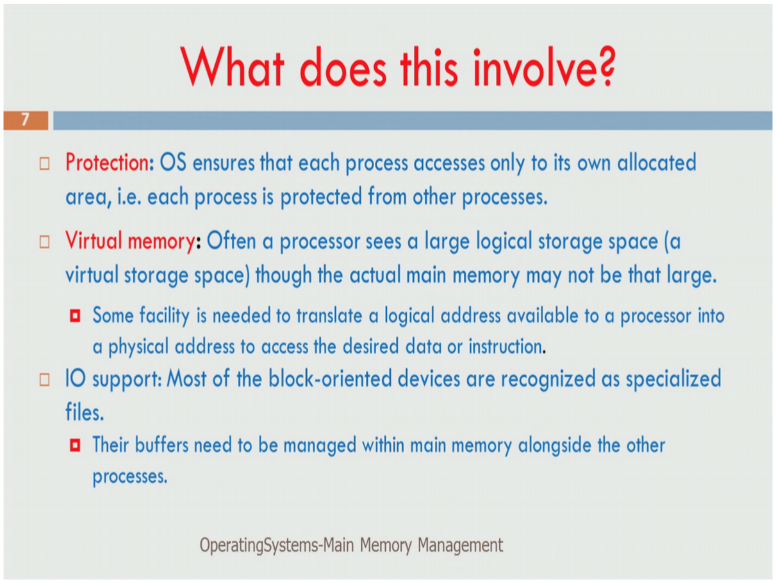
6

- Allocation:-Programs need to be allocated some space.
- Swapping, fragmentation and compaction:-
  - The program moves out from main memory as it terminates.
  - When it moves in it creates an empty space(hole).
    - This is a contiguous un-used area.
  - A new process that moves in may occupy it.
  - There is a possibility that main memory may have too many small holes.
  - This may lead to a situation where new process may not fit these.
  - This means memory is too fragmented.
  - OS re-allocates the existing programs in contiguous regions and creates a large enough free area for allocation to a new process.
    - This is called compaction
- Garbage collection: In order to effectively use holes created by programs that dynamically create them (because they use dynamic data structures) an upper limit to holes created is put. These holes are referred as garbage.
  - Compacting memory is done by collecting free space after these dynamic programs terminate.

OperatingSystems-Main Memory Management

Different aspects of operating systems , so when I want to do memory management I need to do allocation, I need to do swapping, I need to do fragmentation , I should handle fragmentation through compaction and when we do this allocation I might have multiple partitions, I might have and when I reclaimed there are issues related to security. All these things we have summarized in so far.

(Refer Slide Time: 20:39)



What does this involve?

7

- Protection: OS ensures that each process accesses only to its own allocated area, i.e. each process is protected from other processes.
- Virtual memory: Often a processor sees a large logical storage space (a virtual storage space) though the actual main memory may not be that large.
  - Some facility is needed to translate a logical address available to a processor into a physical address to access the desired data or instruction.
- IO support: Most of the block-oriented devices are recognized as specialized files.
  - Their buffers need to be managed within main memory alongside the other processes.

OperatingSystems-Main Memory Management

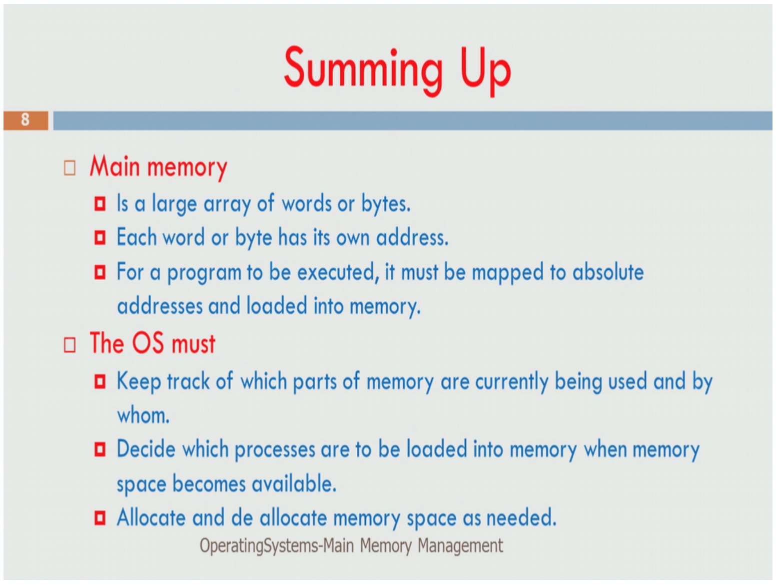
So to achieve this already, I have talked about protection, protection between processes protection across processes for this the operating system takes the help of the architecture we have talked about that in information security 2 course in a very big way. The other important thing is of course, what we call as virtual memory. Virtual memory is that to the user I give, so suppose I at 32 bit architecture to the user we give 4 GB address space, we tell the user you write program that can be as large as 4 GB, but what we do is that we execute that 4 GB program, actually the operating system ensures that that 4 GB program can be executed on a machine (())(21:24), which has just 2 GB of RAM. So the operating system gives the user a virtual memory of 4 GB, but underlying it has only physical memory of 2 GB and so it executes that 4 GB program on the 2 GB system. So how does this happen? The architectural support for this was covered in information security 2 course?

Now we will see some of the operating system perspective also from this, okay and of course, the operating system should also support IO and the IO actually happens as memory mapped IO. So if you look at any of the operating systems for say if I take USB device, there will be a set of memory locations allocated to this USB device. So anything that I have write into the USB device or any a transaction that I do through the USB device can be done through controls pass through this memory locations, the memory mapped IO we call it as memory mamed (())(22:18) IO and the devices can also talk to the memory independent of the CPU and that is also called as direct memory access.

So the operating system should ensure that direct memory access happens without any interference with the CPU in the sense that if there memory allocated for direct memory access that should not be read or accessed by the CPU when the DMA actually happens and also, the operating system should give some set of locations for every device so that the CPU can talk to that device, typically to control it and to see how these operations are happening on the IO devices by actually writing into those memory addresses.

So for every device there is some memory locations allocated those are all really on the RAM, but if that CPU writes into those locations essentially that write will go to the device. So this is actually called as memory mapped IO and we had discussed some amount of this in the information security 2 course.

(Refer Slide Time: 23:30)



Summing Up

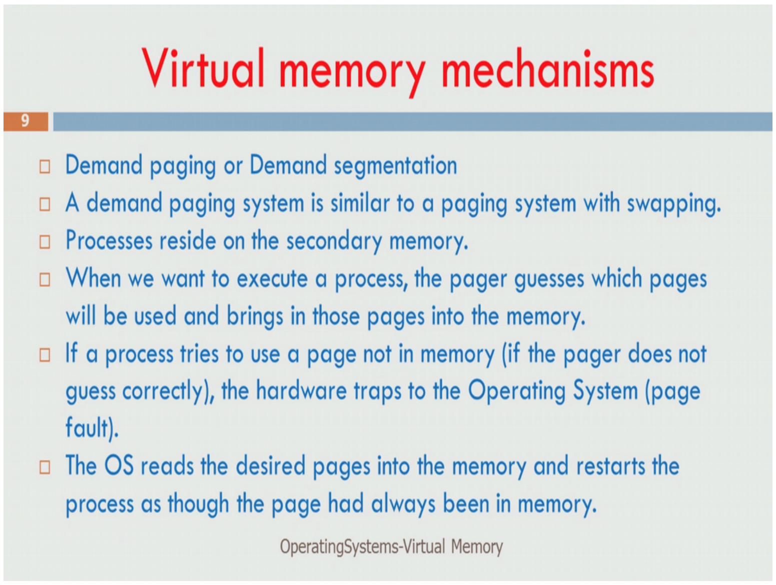
8

- Main memory
  - Is a large array of words or bytes.
  - Each word or byte has its own address.
  - For a program to be executed, it must be mapped to absolute addresses and loaded into memory.
- The OS must
  - Keep track of which parts of memory are currently being used and by whom.
  - Decide which processes are to be loaded into memory when memory space becomes available.
  - Allocate and de allocate memory space as needed.

OperatingSystems-Main Memory Management

So to sum up the main memory is a large array of word or bytes and each word or byte has its own address and for a program to be executed, it must be mapped to absolute addresses are loaded into this memory and the OS must keep track of which parts of the memory are currently being used and by whom and decide which processes are to be loaded into memory when memory space become available and allocate and de-allocate memory space has needed.

(Refer Slide Time: 24:02)



Virtual memory mechanisms

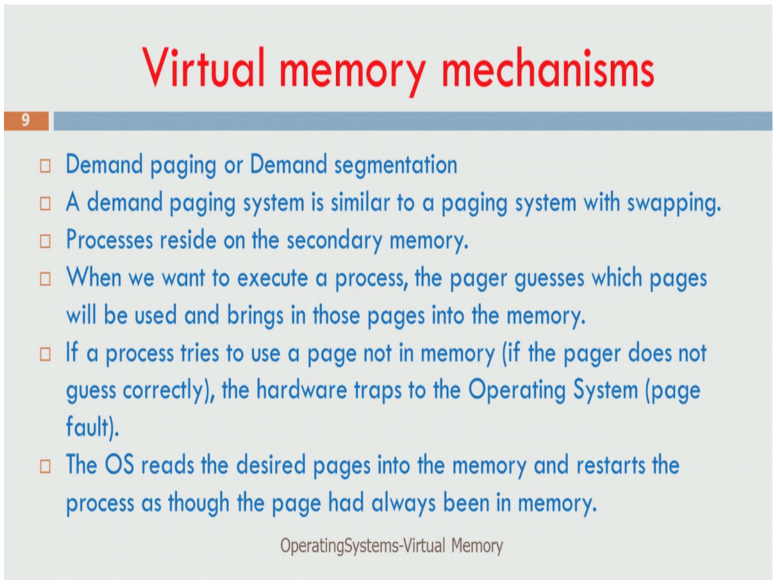
9

- Demand paging or Demand segmentation
- A demand paging system is similar to a paging system with swapping.
- Processes reside on the secondary memory.
- When we want to execute a process, the pager guesses which pages will be used and brings in those pages into the memory.
- If a process tries to use a page not in memory (if the pager does not guess correctly), the hardware traps to the Operating System (page fault).
- The OS reads the desired pages into the memory and restarts the process as though the page had always been in memory.

OperatingSystems-Virtual Memory

Now coming to the virtual memory we have convert the all the architecture aspect in the previous course. Now there something call demand paging or we can also called demand segmentation.

(Refer Slide Time: 24:19)

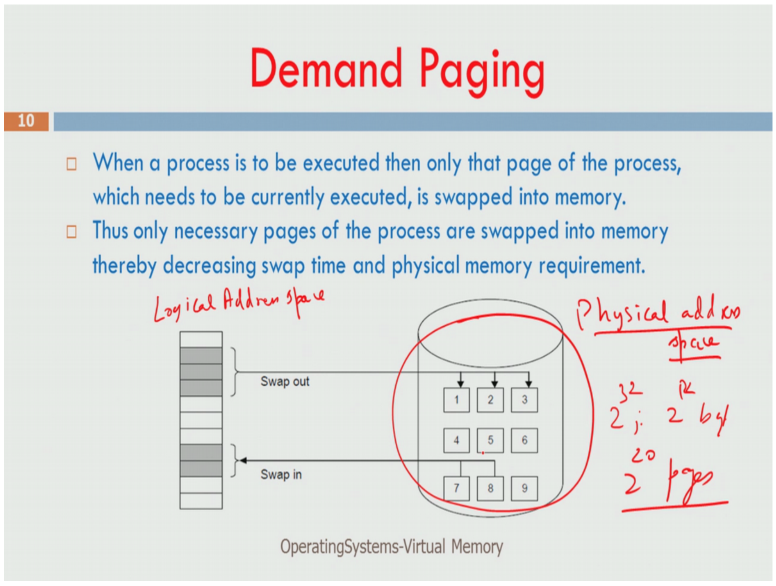


Virtual memory mechanisms

9

- Demand paging or Demand segmentation
- A demand paging system is similar to a paging system with swapping.
- Processes reside on the secondary memory.
- When we want to execute a process, the pager guesses which pages will be used and brings in those pages into the memory.
- If a process tries to use a page not in memory (if the pager does not guess correctly), the hardware traps to the Operating System (page fault).
- The OS reads the desired pages into the memory and restarts the process as though the page had always been in memory.

OperatingSystems-Virtual Memory



Demand Paging

10

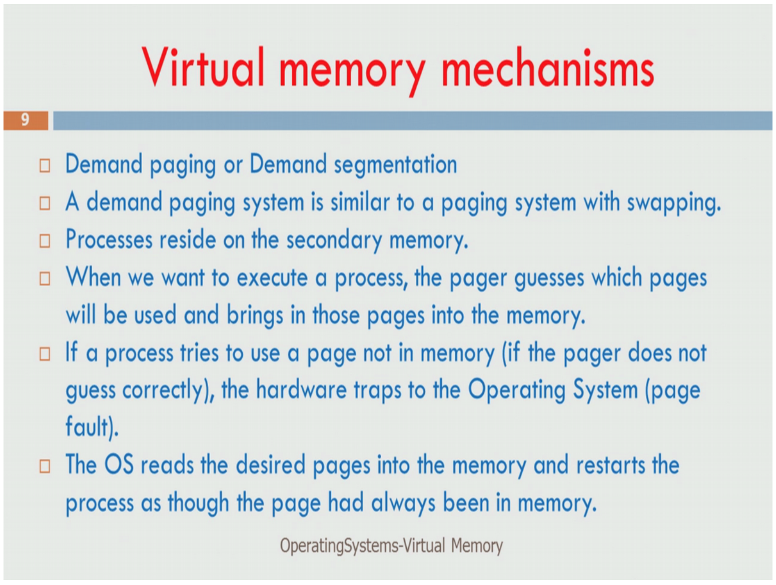
- When a process is to be executed then only that page of the process, which needs to be currently executed, is swapped into memory.
- Thus only necessary pages of the process are swapped into memory thereby decreasing swap time and physical memory requirement.

OperatingSystems-Virtual Memory

So what is demand paging we can very quickly see here in this next slide when a process is too be executed. So we let us call this as the logical address space and this is the physical address space. Now the pages are basically stored in the hard disc. Now as an when I need something to execute I move from the hard disc into the main memory, one page executed and after I do not needed I then swap it out back to the physical space or physical storage space. So if I look at 2 power 32 address space as we have seen in the previous course and if each page is 2 power 12 bytes.

So I have 2 power 20 pages, all the 2 power 20 pages shift if it all they are going to be there will be stored on the disc and the as an when I require a page for execution, I will fetch it from the disc into the main memory execute and the after I do not need it, I then swap out into the main memory into the disc. So this is what we call as demand paging on demand whenever I need something, I bring it from the disc execute and basically transfer it back into the disc. So this is called demand paging.

(Refer Slide Time: 25:56)

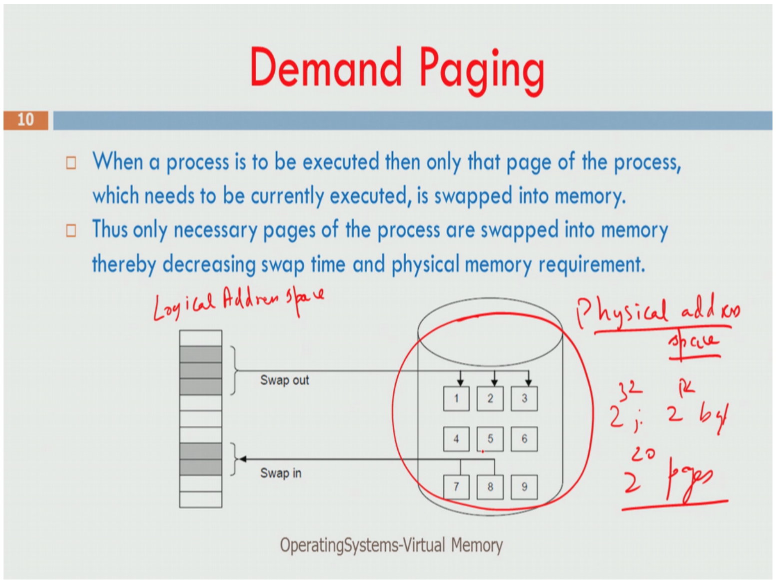


Virtual memory mechanisms

9

- Demand paging or Demand segmentation
- A demand paging system is similar to a paging system with swapping.
- Processes reside on the secondary memory.
- When we want to execute a process, the pager guesses which pages will be used and brings in those pages into the memory.
- If a process tries to use a page not in memory (if the pager does not guess correctly), the hardware traps to the Operating System (page fault).
- The OS reads the desired pages into the memory and restarts the process as though the page had always been in memory.

OperatingSystems-Virtual Memory



Demand Paging

10

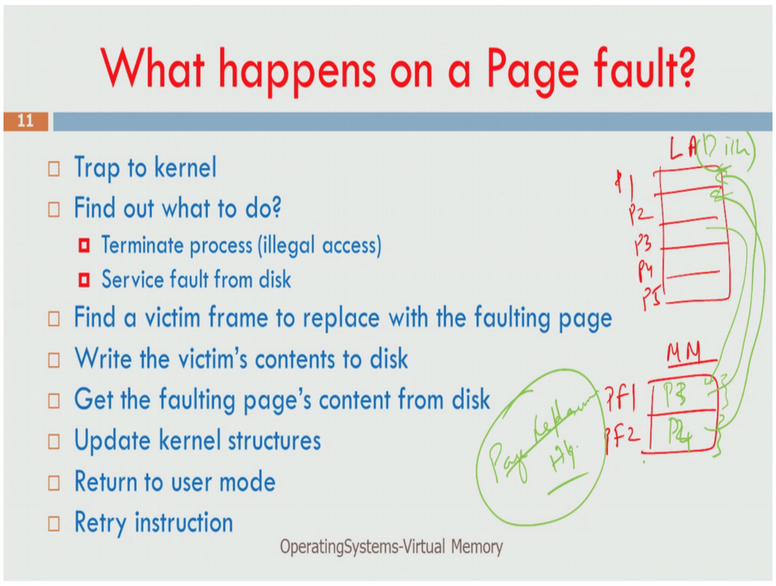
- When a process is to be executed then only that page of the process, which needs to be currently executed, is swapped into memory.
- Thus only necessary pages of the process are swapped into memory thereby decreasing swap time and physical memory requirement.

OperatingSystems-Virtual Memory

So the main issue here is that if the OS reads the desired pages into the memory and restarts the process as through the page had always been in memory. So that is most important thing. So every time when a process is executing whatever it needs to execute the next instruction to be executed the data needed by the next instruction should always be in a memory. So as by von Numan (())(26:16) model my memory will have the necessary data and instruction for execution.

Once that is over are some other process wants to occupy memory we swap these pages back into the hard disc and again when this process comes back into execution we bring it back from the main memory so that from the process perspective it is something that it actually occupy this memory went back to this can came back, but it should restart from exactly where it left and that context is basically saved here. So this is basically what we called as demand paging.

(Refer Slide Time: 27:00)



What happens on a Page fault?

11

- Trap to kernel
- Find out what to do?
  - Terminate process (illegal access)
  - Service fault from disk
- Find a victim frame to replace with the faulting page
- Write the victim's contents to disk
- Get the faulting page's content from disk
- Update kernel structures
- Return to user mode
- Retry instruction

OperatingSystems-Virtual Memory

LA(Disk)
P1
P2
P3
P4
P5
MM
PF1
PF2
P3
P4
Page fault handler

So what happens? How does demand paging essentially works? So let me say that so we will just give you a very simple example here. So let me say that a particular program as say 5 pages. This is the logical address space, but say we are only allocated say 2 pages in the main memory for it to execute. So let me call it as 2 page frames in the main memory as I had explained a photos (())(27:29) inside a photo frame. Similarly, a page will seat into a page frame, so the main memory will have page frames on which you can bring in any page and throw it back.

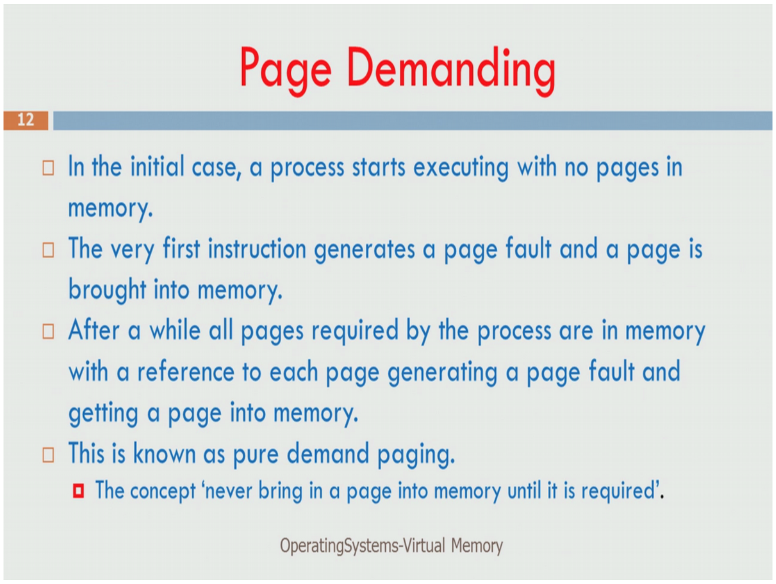
So let us see, how this is going to work. Now when the programs starts executing that is nothing is there in the main memory. So it will now see whether it needs to execute something from page 1, but page 1 is not in the main memory. So immediately a page fault will be created. Once the page fault is created, there is page fault handler which will bring page 1 into main memory.

Now this entire page one will get executed, now so now it is now the program wants to execute page 2, so page 2 brought into the main memory, right. Now the program wants page

3, but it has only 2 page frames, right. So which page to replace, so there comes the concept of what we call as page replacement algorithms. So I go and replace say page 3 page 1 I want to throughout, so my page replacement algorithm says that let us remove page 1. So I swap page 1. So I write back page 1 back here and I bring page 3 here.

Now page 3 executes now I want page 4, now the operating system says now let us remove page 2, so I swap back page 4 page the page 2 and bring in page 4 and so on. So as an when I require new pages, I bring it from the logical the from the disc. So this is disc and this is the main memory, so I move from the disc into the main memory and then you know write back into the disc. So swapping and swap out from the disc to the main memory happens here and this swapping in and swapping out is done by the operating system. So this is another very important thing on what happens on a page fault.

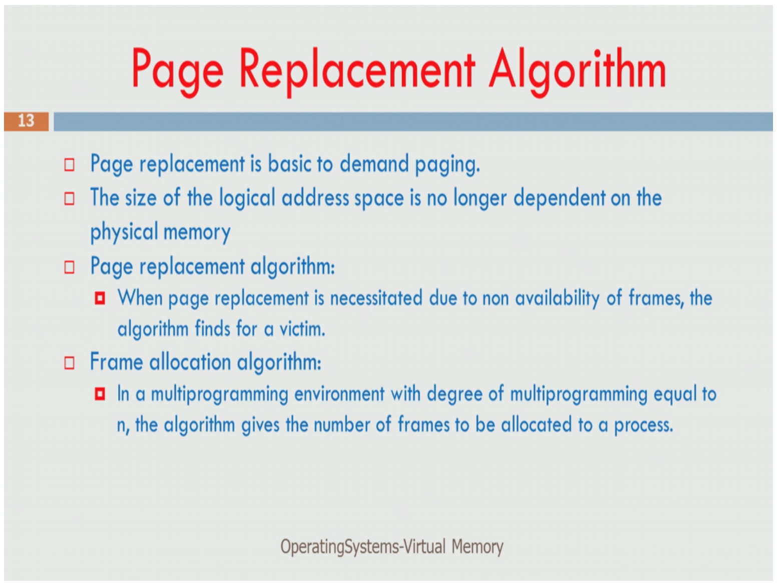
(Refer Slide Time: 29:33)


Page Demanding

12

- In the initial case, a process starts executing with no pages in memory.
- The very first instruction generates a page fault and a page is brought into memory.
- After a while all pages required by the process are in memory with a reference to each page generating a page fault and getting a page into memory.
- This is known as pure demand paging.
  - The concept 'never bring in a page into memory until it is required'.

OperatingSystems-Virtual Memory



Page Replacement Algorithm

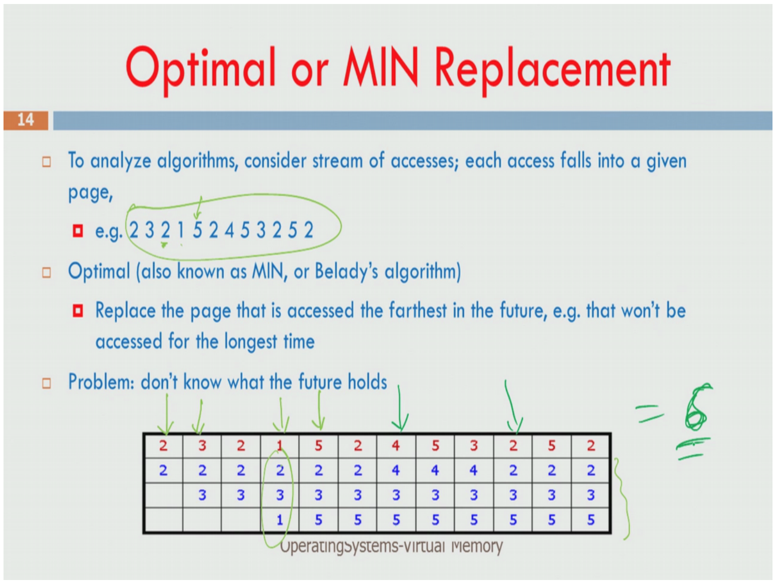
13

- Page replacement is basic to demand paging.
- The size of the logical address space is no longer dependent on the physical memory
- Page replacement algorithm:
  - When page replacement is necessitated due to non availability of frames, the algorithm finds for a victim.
- Frame allocation algorithm:
  - In a multiprogramming environment with degree of multiprogramming equal to n, the algorithm gives the number of frames to be allocated to a process.

OperatingSystems-Virtual Memory

And what we have seeing is that there is need for a page replacement algorithm and the page replacement is very very important which page to replace is determines the actual performance of the program if I have very bad page replacement algorithm please note that the program can essentially execute very slow. So this is something that we need to look at, if I go and touch the paging aspect of the operating system, if I go and change something there, if I could manage to go and do something there, I could make the execution of all the programs very very bad, I can slow down the execution. So this can even be treated as vulnerability, so if somebody wants to attack your system and create problem for you in terms of service time, right. One can go and look at paging any of these paging, so memory allocation or replacement or swapping if they go and touch something there then this can essentially result in your programs getting executed very slow.

(Refer Slide Time: 30:44)



Optimal or MIN Replacement

14

- To analyze algorithms, consider stream of accesses; each access falls into a given page,
  - e.g. 2 3 2 1 5 2 4 5 3 2 5 2
- Optimal (also known as MIN, or Belady's algorithm)
  - Replace the page that is accessed the farthest in the future, e.g. that won't be accessed for the longest time
- Problem: don't know what the future holds

2	3	2	1	5	2	4	5	3	2	5	2
2	2	2	2	2	2	4	4	4	2	2	2
	3	3	3	3	3	3	3	3	3	3	3
			1	5	5	5	5	5	5	5	5

= 6

OperatingSystems-Virtual Memory

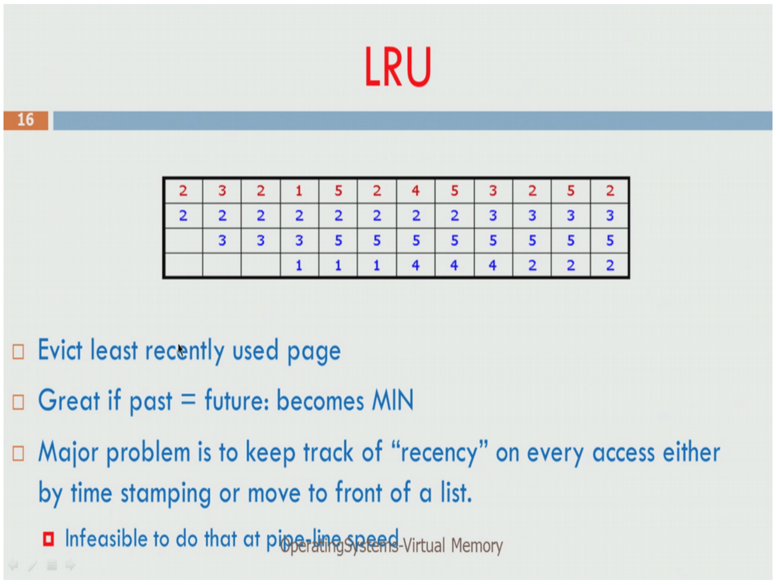
Let us look at some of these aspects of this. So let us look at see there is program which has say 5 pages and this is how it is going to access these pages, right, so that program is allocated 3 page frames, right. So first it the it looks for 2, 2 is not there, so this is page fault. So it brings 2 inside, second it looks for 3, it brings 3 inside, the third again it is accessing 2, but 2 is already there in the memory, so there is no page fault then now it is asking for 1. So 1 is brought inside okay. So this is a page fault.

Now it is trying to access 5, after some point is trying to access 5. Now which one to replace should I replace 2 or 3 or 1, I decide to replace 1, so this is a page fault. Now it is looking for 2, now there is no page fault see, please note that if I had replaced 2 with 5 rather than 1 with 5, this would have resulted in a page fault. So if I had replaced this 2 with 1and sorry, this replace with 5 and retain this 1, this would have caused a page fault.

Now since I did not do this, I get I do not get page fault here. Now 4 comes in, so which 2 replace I am replacing it with 2 I am replacing 2. So this is a page fault. Now 5 comes in please note that if I have replace 5 then this would have caused a page fault, so 5 is not a page fault for me 3 comes in its not a page fault. Now again 2 comes in, so I replace 4 again 5 comes in there is no page fault there 2 (())(32:54). So this basically as resulted in 5 page fault (())(32:58) 1, 2, 3, 4, sorry 6 page faults, every page fault essentially means writing 1 page and reading from another page form the disc and disc is very very slow.

So more the number of page faults slower is my algorithm, so is my execution time. So I need to reduce the number of page faults and that is what basically the page replacement algorithm does (())(33:23).

(Refer Slide Time: 33:25)



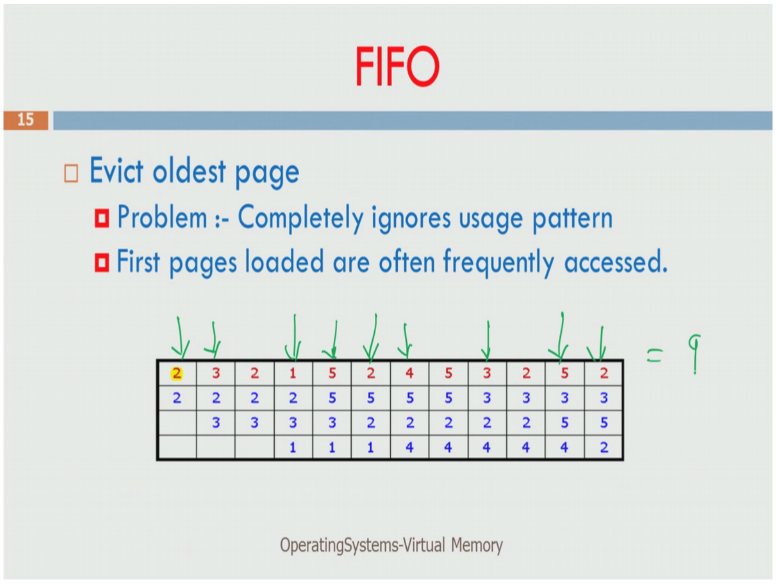
LRU

16

2	3	2	1	5	2	4	5	3	2	5	2
2	2	2	2	2	2	2	2	3	3	3	3
	3	3	3	5	5	5	5	5	5	5	5
			1	1	1	4	4	4	2	2	2

- Evict least recently used page
- Great if past = future: becomes MIN
- Major problem is to keep track of "recency" on every access either by time stamping or move to front of a list.
  - Infeasible to do that at pipe-line speed

OperatingSystems-Virtual Memory



FIFO

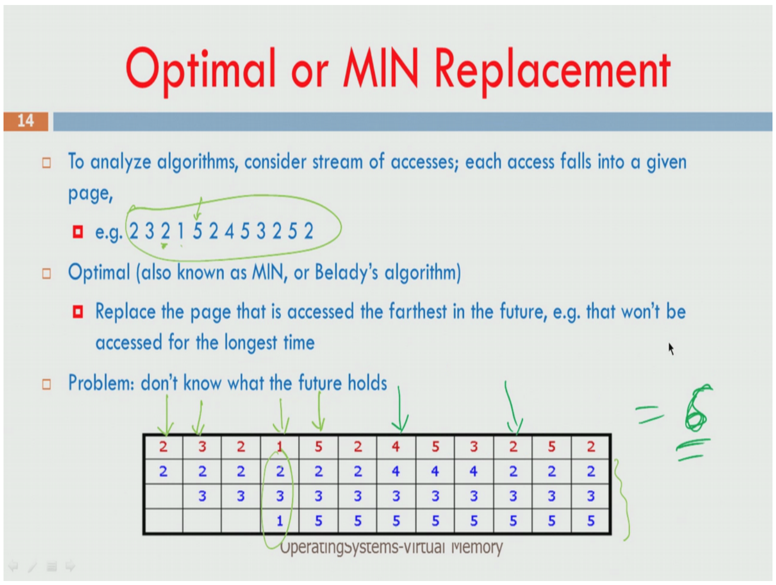
15

- Evict oldest page
  - Problem :- Completely ignores usage pattern
  - First pages loaded are often frequently accessed.

2	3	2	1	5	2	4	5	3	2	5	2
2	2	2	2	5	5	5	5	3	3	3	3
	3	3	3	3	2	2	2	2	2	5	5
			1	1	1	4	4	4	4	4	2

= 9

OperatingSystems-Virtual Memory



Optimal or MIN Replacement

14

- To analyze algorithms, consider stream of accesses; each access falls into a given page,
  - e.g. 2 3 2 1 5 2 4 5 3 2 5 2
- Optimal (also known as MIN, or Belady's algorithm)
  - Replace the page that is accessed the farthest in the future, e.g. that won't be accessed for the longest time
- Problem: don't know what the future holds

2	3	2	1	5	2	4	5	3	2	5	2
2	2	2	2	2	2	4	4	4	2	2	2
	3	3	3	3	3	3	3	3	3	3	3
			1	5	5	5	5	5	5	5	5

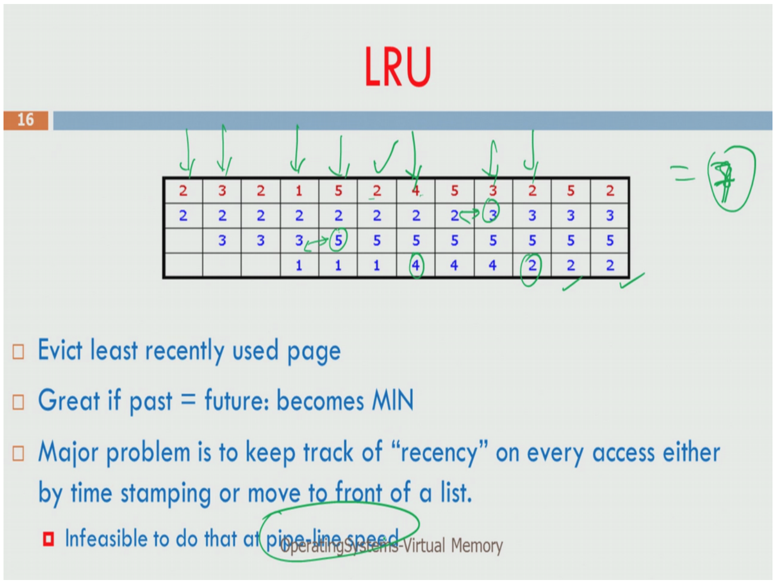
$= \underline{\underline{6}}$

OperatingSystems-Virtual Memory

Now one of the interesting algorithm is FIFO that is evict the old page first in first out. This we would have covered in basic operating system courses. So when I look at FIFO first there is page there is page fault to start with that is 2 then 3 gives the page fault then there is no page fault here 1 gives another page fault. Now 5 comes in since 2 is the oldest page, 2 gets replaced by 5 and that is page fault again.

Now again 2 comes in out of 5, 3 and 1, 3 is the oldest to enter, so that gets replaced. Now 4 gives me a page fault now 5 there is no page fault. Now 5 is the oldest as of now when 3 comes in 5 gets replaced 2 there is no page fault. Now again when 5 comes in, 2 is the oldest fellow to enter approach that it is out, again 2 comes in so this is older. So what happen 9 page faults happens, so actually we could have executed with 6 we had way of doing it with 6, but now I see 9, right. So FIFO **is** is not so good for this particular program.

(Refer Slide Time: 34:48)



LRU

16

2	3	2	1	5	2	4	5	3	2	5	2
2	2	2	2	2	2	2	2	3	3	3	3
	3	3	3	5	5	5	5	5	5	5	5
			1	1	1	4	4	4	2	2	2

= 7

- Evict least recently used page
- Great if past = future: becomes MIN
- Major problem is to keep track of "recency" on every access either by time stamping or move to front of a list.
  - Infeasible to do that at pipe-line speed

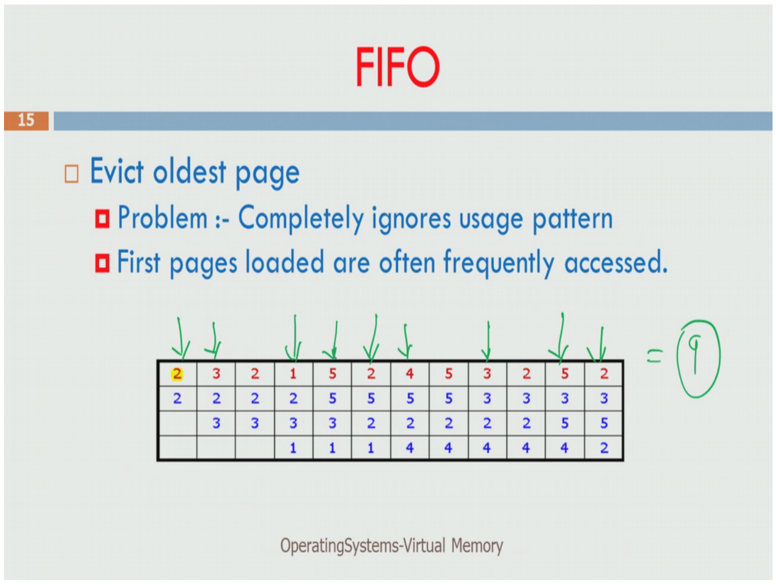
Operating Systems-Virtual Memory

The next one is what we call as LRU the least recently used. So what is least recently used? So let us start this, let us very quickly go through this, 2 is here, 3 is here again 2 is used then 1, so this is not a page fault, but these 3 are page faults 2, 3 and 1. Now 5 comes in out of 2, 3 and 1 the least recently use this 3, because 1 and 2 have been used much recently before 3. So 5 actually goes and replace 3. Now 2 is still there (())(35:38), 4 is out of 2, 5 and 1, please note that at this point 2 and 5 are more recently used in one, so 4, 4 replaces 1.

Now 5 is still there, now between 2, 5 and 4, 2 is least recently used. So when 3 comes in 3 replaces 2 then between 3, 5 and 4, please note that 2 is 4 is least recently used when 2 comes is still there and the 3, 5, 2 works and 3, 5, 2 works, okay. So total no of this is 8, 8 page faults or now sorry it is 7 page faults, okay because this is not, so this is not a page fault. So totally 7 page faults happen here, okay. So this is least recently used, but then when I want to implement least recently used please note that we need to have information about this uses I need to consistently mention.

So when I say, so 5, 5 needs to occupy which one to replace between 2, 3 and 1 I should know that 3 has been least recently used. So I need to maintain some information about the history of usage and that makes this LRU implementation much complex. That is we say it is infeasible to do this at pipeline speed right at very fast speed.

(Refer Slide Time: 37:26)



FIFO

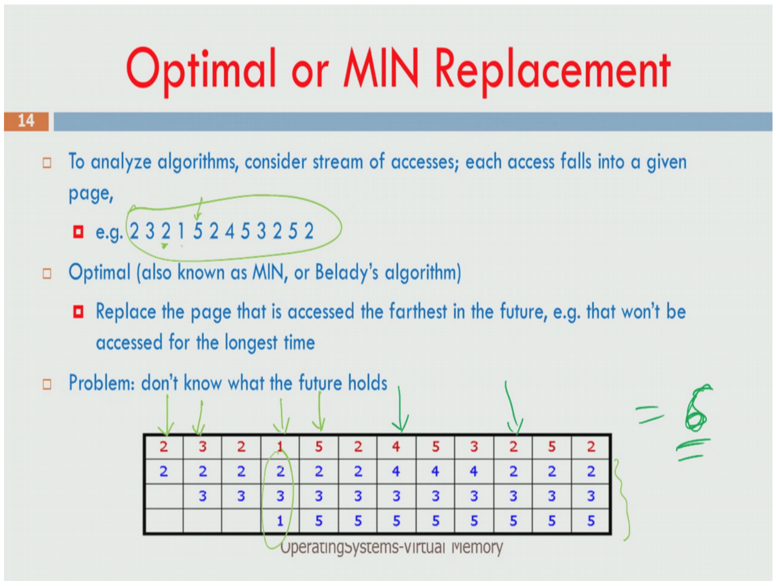
15

- Evict oldest page
  - Problem :- Completely ignores usage pattern
  - First pages loaded are often frequently accessed.

2	3	2	1	5	2	4	5	3	2	5	2
2	2	2	2	5	5	5	5	3	3	3	3
	3	3	3	3	2	2	2	2	2	5	5
			1	1	1	4	4	4	4	4	2

= 9

OperatingSystems-Virtual Memory



Optimal or MIN Replacement

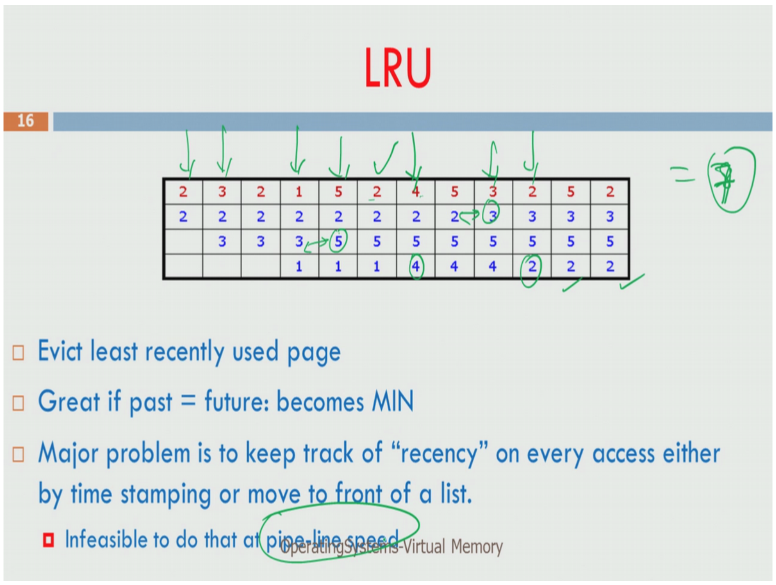
14

- To analyze algorithms, consider stream of accesses; each access falls into a given page,
  - e.g. 2 3 2 1 5 2 4 5 3 2 5 2
- Optimal (also known as MIN, or Belady's algorithm)
  - Replace the page that is accessed the farthest in the future, e.g. that won't be accessed for the longest time
- Problem: don't know what the future holds

2	3	2	1	5	2	4	5	3	2	5	2
2	2	2	2	2	2	4	4	4	2	2	2
	3	3	3	3	3	3	3	3	3	3	3
			1	5	5	5	5	5	5	5	5

= 6

OperatingSystems-Virtual Memory



LRU

16

2	3	2	1	5	2	4	5	3	2	5	2
2	2	2	2	2	2	2	2	3	3	3	3
	3	3	3	5	5	5	5	5	5	5	5
			1	1	1	4	4	4	2	2	2

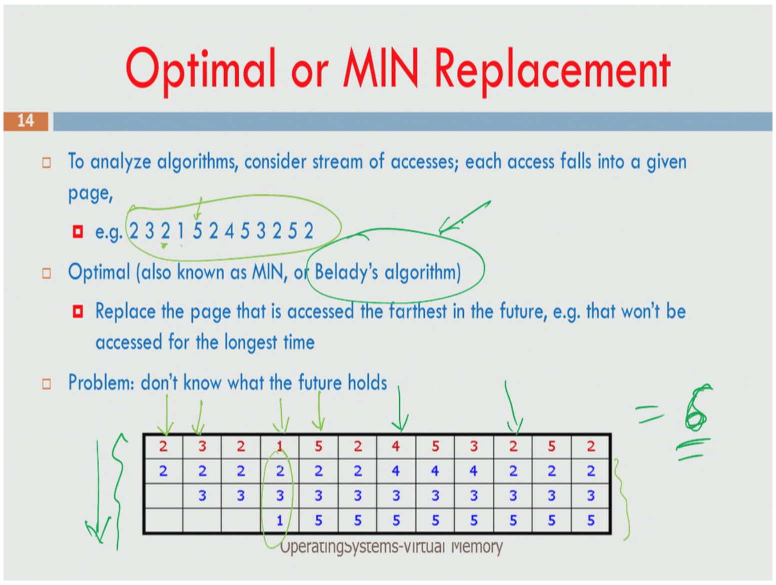
= 7

- Evict least recently used page
- Great if past = future: becomes MIN
- Major problem is to keep track of "recency" on every access either by time stamping or move to front of a list.
  - Infeasible to do that at pipe-line speed

OperatingSystems-Virtual Memory

So out of the 3 algorithms we have seen so far, the first one took 6, second one took 9, third one took 7, so 6 is the best, but please note that we do not have an algorithm for 6 we just did some optimal 6 is optimal here we can prove that, but we just took an arbitrary decision to replace something here but it (())(37:46) we did not tell you what the algorithm here is, actually we do not have an algorithm which can give 6, like FIFO and LRU gives you that 9 and 7, okay.

(Refer Slide Time: 37:58)



Optimal or MIN Replacement

14

- To analyze algorithms, consider stream of accesses; each access falls into a given page,
  - e.g. 2 3 2 1 5 2 4 5 3 2 5 2
- Optimal (also known as MIN, or Belady's algorithm)
  - Replace the page that is accessed the farthest in the future, e.g. that won't be accessed for the longest time
- Problem: don't know what the future holds

2	3	2	1	5	2	4	5	3	2	5	2
2	2	2	2	2	2	4	4	4	2	2	2
	3	3	3	3	3	3	3	3	3	3	3
			1	5	5	5	5	5	5	5	5

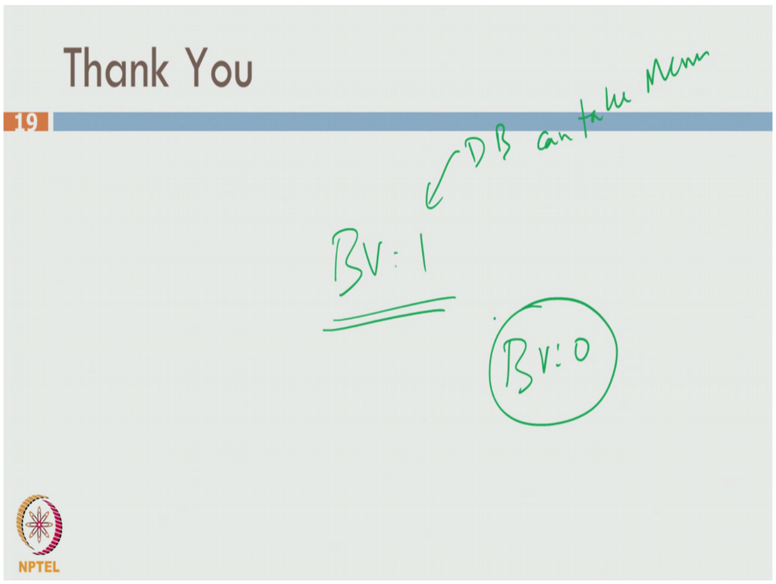
= 6

OperatingSystems-Virtual Memory

Now one very important thing about cache placement is the Beladys anomaly, so how can I reduce the number of page faults now it is 6 , how can I reduce it if I increase the no of page frames that is one intuitively that looks (())(38:14) if I increase the number of page frames per a process then the page fault I give more memory that means page fault will reduce, but that is not the case , there is Beladys anomaly algorithm which you learn in basic operating system course, which basically tells you that as I keep increasing the page frame my no of page fault can also increase rather than decreasing.

So I suggest that you look into basic operating systems course lectures or books to understand what is Beladys anomaly, okay. So we with this we end the module 9 which is memory management 1and what the takeaway from this module is that memory management is crucial and if we go and look at some of the memory allocation page allocation algorithms and if we could have an access into the swapping mechanism as a hacker if I go and look at the hacking either swapping mechanism or allocation policies and if I can go and do something there then we can really go and bring the system down.

(Refer Slide Time: 40:39)



Thank You

19

DB can take Mem
BV: 1
BV: 0
NPTEL

Very important thing is that if you look at some of the operating systems they have certain boot time variables boot time variable or the variables that or considered by the system when at the boot time. The boot time variable can give permissions say to the data base to basically you know allocates memories for itself, right okay, allocate sheared memory for itself. So suppose I have data base running on in a in operating system, now the operating system can basically give permission to the database to allocates its own memory.

Suppose this system exist, so somebody so what would happen is that you boot a system at some point of time and the system is actually running then and it is maintain by an employ of a company and suddenly the company decides to throw him off. So the employ actually gets (())(40:20), so what he needs to do is to go to that variable before just these quitting you know the tomorrow they are going to send him up today you may have access you can go and make the boot time variable say let us say that the boot time variable also 1 so that Bv is one means data base can take memory, so and this decided at boot time after that if you change to zero nothing will happen. So descrintal (())(40:53) employ can make this zero on the day he is quitting and move off. Now you are not going to shut down the server say for next 5 or 6 months somewhere you go and do the check at that point you go and reboot the server, on the day when you reboot this Bv will be zero. So what would happen is?

Now the data base cannot actually do its own memory allocation. So the database performance will come down then there is no way by which you will know that, because you and then after it actually you can take lot of time to find out that this boot variable is the reason for that you may now start going to the data base vender and he tell you okay we need by more licenses by more (())(41:41) that they will make more business hardware will the fellow will send tell you more hardware data base will sell you more license. So (())(41:48) actually it write more money and then finally you will realize that this boot variable as becomes zero.

Now who it would be rather 7 to 8 months before which, it has been actually met to zero, but you will only detect it 8 months later. So this is these are some very interesting security vulnerabilities that we need to be aware off. So when the operating system is booted if there is a control even to admin to go and handle the swap or go and change the swap policies etcetera then it can basically lead to lot of vulnerabilities. So that is one very important thing that we need to learn from the memory management point of view, we will continue on this in the memory management 2 module, thank you.