

**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**  
**Mod02 Lecture 10**  
**Module 10: Memory Management-2**

So welcome to module 10, so now we deal it with the memory management perspective. One of the important things that we need to keep in mind is that we are now looking at servers. So as this course also titles is a server management when we look at servers, there will be multiple processes that will be executing on the server and we call it as degree of multiprogramming, right. What (0:38) degree of multiprogramming? The number of processes that are simultaneously ready to be executed on the CPU. So each of these processes would be allocated some memory and on the logical address space and they have to whenever they need to execute, they need to go to the main memory and execute that the corresponding pages have to be move to the main memory should be available in main memory and we have to start executing.

So suppose I have 10 processes the at least 10 pages should be or 10 or 20 pages should be available in the main memory. So each process will be occupying say one or two pages etcetera. Now what would happen is that if I keep on increasing, say suppose I have 20 pages and I have 10 processes then I can allocate 2 pages for each process, suppose I have now I make it 20 processes that means I am increasing the degree of multi programming then each process will get on an average only one page to execute, right. So as I keep increasing number of processes that could simultaneously exist execute on a system, what I call as degree of multi-programming, which I increasing then the number of page faults will increase that means then the CPU will be spending more time handling page faults rather than doing any constructive execution of the process and this state is actually called as thrashing, right.

(Refer Slide Time: 2:23)

## No of Page Faults vs Page Allocation

19

- Desired behavior of paging algorithm: reduce page fault rate below “acceptable level” as number of available frames increases
  - Q.: does increasing number of physical frames always reduce page fault rate?
  - A.: usually yes, but for some algorithms not guaranteed (“Belady’s anomaly”)

OperatingSystems-Virtual Memory

So the nectar beyond some level actually becomes poison, so if we keep on increasing the degree of multi-programming somewhere there will be a heat and that heat comes in the form of what we call as thrashing here. So this is how the thing, so what we see on your right hand side is the page fault rate with the number of frames allocated, right. So there is a desired behavior of paging algorithms is to reduce page fault rate below acceptable level as number of available frames increases, right that means, so the question here is, does increasing number of physical frames always reduce page fault rate? Actually we have said in the previous class previous module that this is usually, yes but for some algorithms, this is not guaranteed and we have said about Beladys anomaly.

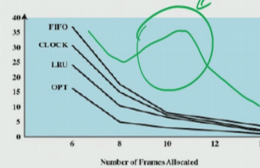
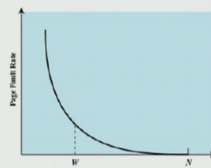
What you see on the left hand side is the page fault rate and y axis is a page fault rate and x axis is the number of frames and as I keep increasing the number of frames, you expect the page fault rate to become zero and for different 4 different algorithms like FIFO, clock, LRU, OPT, OPT is the optimal most optimal algorithm. now we see as the number of frames is increasing the page fault actually decreases, but there are some algorithms where which suffers from this Beladys anomaly where as the number of frames allocated increases, the page fault can also increase, for example, it may be something like this, but then it can go up and then come down. So this is what we call as this Beladys anomaly, right.

(Refer Slide Time: 3:49)

## No of Page Faults vs Page Allocation

19

- Desired behavior of paging algorithm: reduce page fault rate below “acceptable level” as number of available frames increases
  - Q.: does increasing number of physical frames always reduce page fault rate?
  - A.: usually yes, but for some algorithms not guaranteed (“Belady’s anomaly”)



OperatingSystems-Virtual Memory

## When Virtual Memory Works Well

20

- Locality
  - 80 % of accesses are to 20 % of pages
  - 80 % of accesses are made by 20 % code
- Temporal locality
- Page that is accessed will be accessed again in near future.
- Spatial locality
  - Pre-fetching pays off : If a page is accessed, neighboring page will be accessed.
- If VM works well average access to all memory is about as fast as access to physical memory

OperatingSystems-Virtual Memory

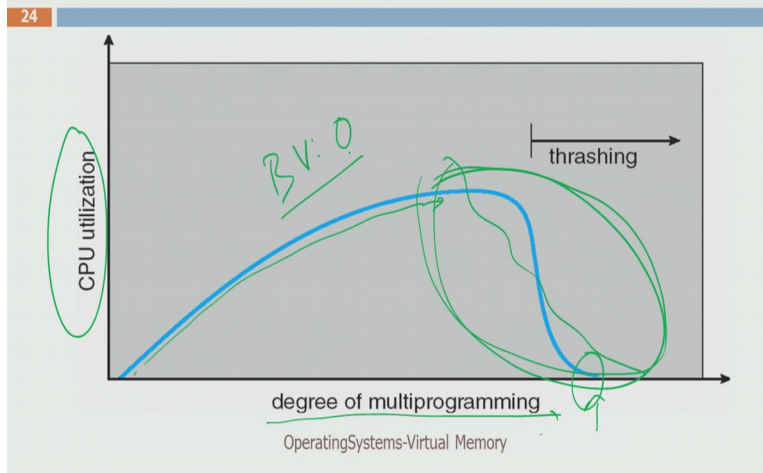
## Why would an O.S Thrash?

23

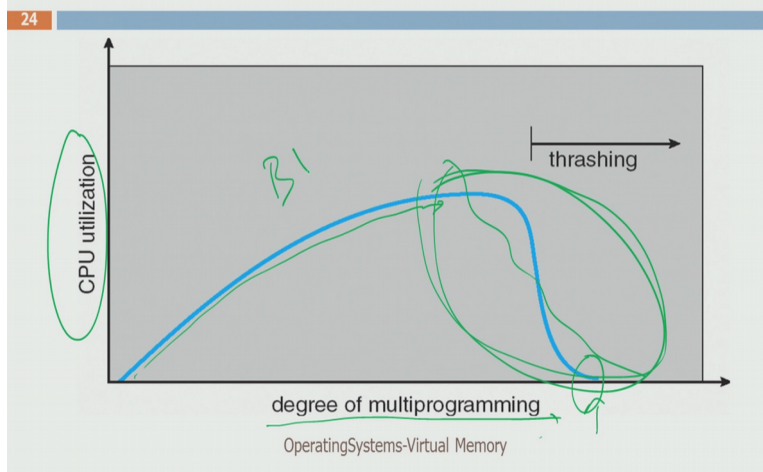
- The operating system closely monitors CPU utilization.
- When CPU utilization drops below a certain threshold, the operating system increases the degree of multiprogramming by bringing in a new process to increase CPU utilization.
- Let a global page replacement policy be followed.
- A process requiring more frames for execution page faults and steals frames from other processes which are using those frames.
- This causes the other processes also to page fault. Paging activity increases with longer queues at the paging device but CPU utilization drops.
- Since CPU utilization drops, the job scheduler increases the degree of multiprogramming by bringing in a new process.
- This only increases paging activity to further decrease CPU utilization.
- This cycle continues.
- Thrashing has set in and through put drastically drops.

OperatingSystems-Virtual Memory

## Thrashing Illustration



## Thrashing Illustration



Now with this as a background that we have seen why paging actually works if I have program say which will spawn across say 100 pages, suppose I have a program, which will spawn across say 100 pages if I give just give it some 20 pages till it will work file (4:03) it will not create lot of page faults, right. The reason is that the 80-20 rule 80 percent of access are only to 20 percent of pages or 90 percent of the program executes for 10 percent of the time and 10 percent of the program executes for 90 percent of the time, this is a 90-10 rule in software engineering so that means, so 80 percent of the access are of only for 20 percent of the pages.

So if those 20 percent is actually inside the our (4:31) memory then your program will start working fast. So these 20 percent of the pages if they are moved from the disc memory and they are available then it will start working very well. Now what does that 20 percent

mean if I have 100 pages then 20 pages should be allocated surely, but if I keep increasing the degree of multiprogramming then what will happen is that, so this is what happens if I start increasing, I will not be in a position to give those 20 percent pages.

So if I have an 100 page program and I have to give 20 pages has an operating system for that to execute, but since there are so many processes there. This is degree of multiprogramming there then what will happen I will not be in a position to give you 20 percent of the pages, so then what happens. So I will give less than 20 percent and then so as a process, I will land up with large more of page faults and so your CPU will start handling more page faults rather than constructive execution and that is why you actually see trend (5:41) of decreasing here this shown.

So as your degree of multi-programming increases, your CPU utilization starts falling down at some point it will actually become zero, the system will starts handling, right. So till that point as my degree of multi-programming increases you see and upward trend (6:02) in the CPU utilization but beyond some point it faults and the reason for this fault is because CPU starts handling more the more of page faults rather than doing constructive execution and the reason why I am getting more page fault is because, for every process at least 20 percent of the most important pages should be there in memory and they are not there in memory and so these processes land up with more page faults essentially making CPU work more for the page fault rather than constructive execution.

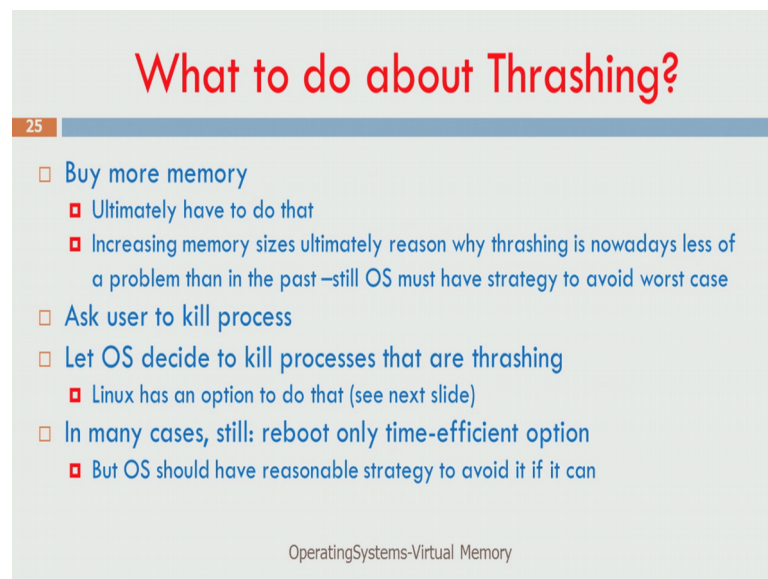
So this is where the thrashing starts, right and that 20 percent of the page is basically call the working set of the process, so always my working set should be in the memory. If my working set of pages the 20 percent of the pages call working set of pages that working set of pages if they are not there in the memory then you will land up with thrashing. So one of the ways by which a person can hack into the system is somehow if he is going to spawn more processes which having a way of spawning more processes by which he could increase the degree of multi programming, some vulnerability exist which could increase the degree of multi programming and these processes can be some junk (7:17) processes too, but if he is going to get to this thing then possibly, he can go and thrash the system, right, for example even in the last before I had winded up the module the previous module, I gave an example of a boot time variable of a server.

So if my boot time variable actually becomes zero in that case that means the data base cannot access allocate its own memory, so then the database now starts allocating request

operating system to allocate memory for it and so the operating system, now in a particular server environment like that, there could be lot of processes along with database as a process. So now the operating system will also treat this database as one of those processes and so it will only allocates some amount of memory to this database, but the database actually is lot more memory because the working set of a database server can be much larger than the working set of a normal process.

Now if the operating system with this boot variable zero if it starts looking at the database process and also the normal process in the same way then what will happen, the database process will land up with more page faults and essentially the database performance goes down, a server or the entire say, for example your core banking in a banking environment or your core insurance in insurance environment, these things will start, behaving very slowly, it can even thrash the system right for. So this is very very important that we need to understand here.

(Refer Slide Time: 8:52)



25

## What to do about Thrashing?

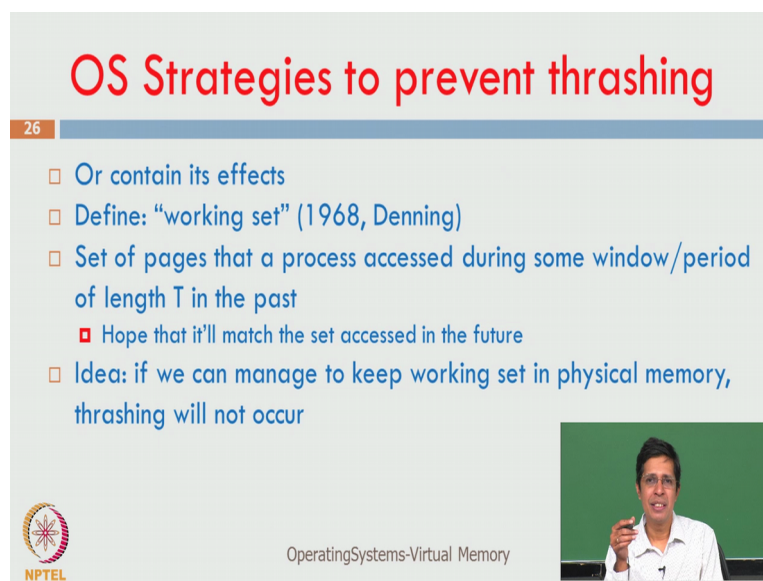
- Buy more memory
  - Ultimately have to do that
  - Increasing memory sizes ultimately reason why thrashing is nowadays less of a problem than in the past –still OS must have strategy to avoid worst case
- Ask user to kill process
- Let OS decide to kill processes that are thrashing
  - Linux has an option to do that (see next slide)
- In many cases, still: reboot only time-efficient option
  - But OS should have reasonable strategy to avoid it if it can

OperatingSystems-Virtual Memory

So what to do about thrashing? So when a thrashing comes you have to go and kill some processes, but most importantly you should see that the thrashing should not recur ((9:00)). So thrashing just by increasing the memory possibly thrashing cannot solve, so suppose a server starts hanging, the immediate reaction would be that there may be memory issue memory based issue. Now that memory issue will never get solved by increasing the amount of memory if some somebody comes and says, you have a problem, so increase the amount of memory, it may solve or it may not solve. The problem can be much more complex than this issue, so that is one thing that we should understand.

So when there is a process that is hanging it is not just because there is less amount of CPU or less amount of memory, it might be that there is something like thrashing that is happening inside, which may increasing the number of CPU or increasing the amount of memory, you cannot stop it, it may be a temporary relief (9:56), but it cannot be permanent relief there the problem which be much more deep rooted then that and that is what we are trying to hint through this stuff. So as a perspective security engineer, you should you should keep these things in mind , because if you go and handle large complex data centers etcetera, these things comes extremely crucial, these type of knowledge is extremely important and crucial.

(Refer Slide Time: 10:22)



The slide is titled "OS Strategies to prevent thrashing" in red text. It features a blue header bar with the number "26" on the left. The main content is a list of points in blue text:

- Or contain its effects
- Define: "working set" (1968, Denning)
- Set of pages that a process accessed during some window/period of length T in the past
  - Hope that it'll match the set accessed in the future
- Idea: if we can manage to keep working set in physical memory, thrashing will not occur

In the bottom right corner, there is a small video inset showing a man in a white shirt speaking. At the bottom left, there is an NPTEL logo. At the bottom center, the text "OperatingSystems-Virtual Memory" is displayed.

So (9:56) I am just going brushing through this slides, because I have explained these slides in great detail, these slides basically talks about what is a working set? This was 1968, this we actually introduced by denning, so very very simple definition of this is set of pages that I process access during some window period of some time period t in the past, okay. So if this working set could be maintained in the main memory then the page fault can basically decrease.

(Refer Slide Time: 10:53)

The slide is titled "Working Set" in red text. Below the title is a blue horizontal bar with the number "27" in white. The main content is a list of ideas in blue text, with some sub-points in red text. The footer is "OperatingSystems-Virtual Memory" in a small, grey font.

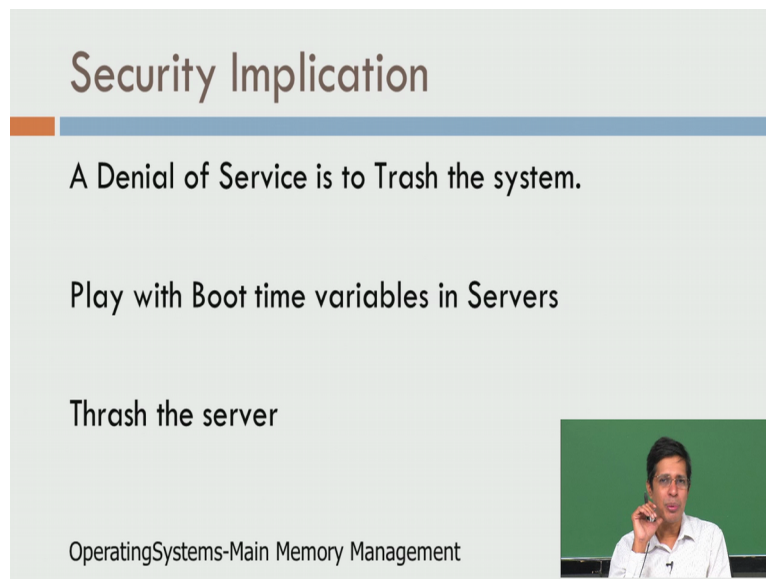
- Suppose we know or can estimate working set –how could we use it?
- Idea 1: give each process as much memory as determined by size of its WS
- Idea 2: preferably evict frames that hold pages that don't seem to be part of WS
- Idea 3: if WS cannot be allocated, swap out entire process (and exclude from scheduling for a while)
  - "medium term scheduling", "swap-out scheduling"
  - (Suspended) inactive vs active processes
  - Or don't admit until there's enough frames for their WS ("long term scheduling")

OperatingSystems-Virtual Memory

So if I have the knowledge of the working set an importantly, if the working set cannot be allocated we swap out entire process and then bring it when you have enough memory to bring the working set, okay. So rather than keeping that process and that creating lot of page fault hindering the other processes, one thing is to keep the process that has as a large working set away or giving it more pages that is what we told for data base it has a large working set, the operating system could give permission to the database process to handles its own memory, so sometimes database is becomes independent of the operating system just needs the permission of the operating system, the operating system gives a huge amount of memory, database takes care of handling the memory, so that is another thing. So these are some of the issues that people have worked out in order to handle this working set problem.



(Refer Slide Time: 11:46)




**Security Implication**

A Denial of Service is to Trash the system.

Play with Boot time variables in Servers

Thrash the server

OperatingSystems-Main Memory Management



So the important takeaway from this point is that if I could increase the degree of multi-programming see that the working set is not in the main memory then I can basically go and you know bring the performance of the system down, essentially causing what we call as denial-of-service and this is one thing that you need to keep in mind as a security engineer as a take away from this course, please again note this is not a course on operating system, this is a course on information security, so I am just trying to give some glimpse of operating system fundamentals, which could have implications in information security, right. Thank you.