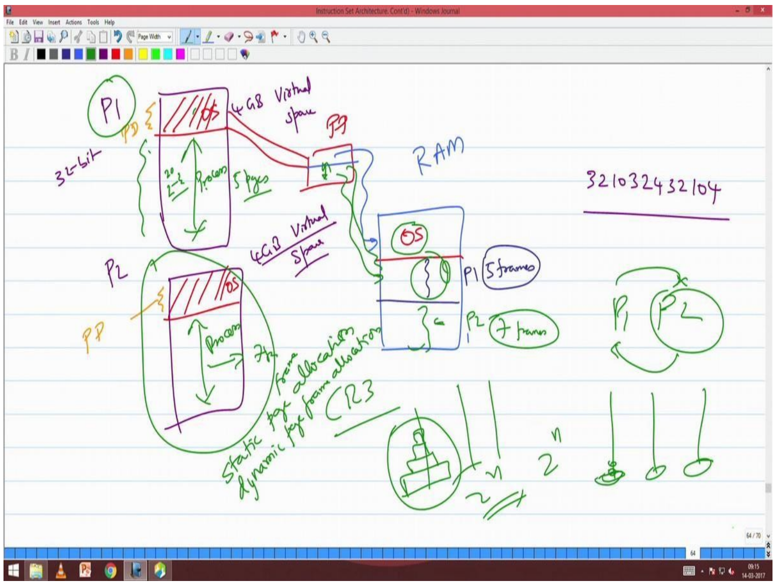
Computer Organization and Architecture

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

Lecture – 30
Page Frame Allocation, Beledy's Anomaly

We were discussing about paging, in the last class and then we are saying that every process can be given its own page table and so this is how it works.

(Refer Slide Time: 00:37)



So, I have process P 1, I have process P 2 each will be given a 4 GB virtual space assuming at 32 bit architecture. And essentially here both of these processes is will need operating system routines I has I explained you last time. So, some part of this will go for the operating system, why it needs operating system why every routine will need every every process will need operating system for basically performing the system calls etcetera right. So, and even your page translation mechanism everything is part of your operating system. So, some part of the virtual address space will be given for this, and the remaining entire space is available for the process while this is OS.

Now, what actually happens is within this part of whatever this part there will be the page directory also mapped. Now in the physical page this is the ram again some part will be used for the OS it will be reserved for the OS. So, when I load when P 1 is executing its page directory. So, let me just zoom that page directory here, its page directory will have some parts which will map onto the OS, and then there will be lot more of parts here, but then what happens is that when P 1 is executing let there will be some k frames all the frames will not be given to P 1.

So, in the physical memory please note carefully in this physical memory says some part will be allocated to P 1 say let us say some 5 frames will be allocated to you 5 page then some part will be allocated to P 2 say again some 7 frames even though P 1 is given the entire 4 GB of address space it has to when P 1 is executing the OS pages are mapped here and there are only some pages all the pages here have to get mapped onto these 5 pages. So, for me to execute this entire program marked in green here I have only 5 pages that are given for me.

So, I have to use say this can be say two power let us say two power 20 pages. So, some are used. So, let me say 2 power 20 minus epsilon pages that many pages are actually available to the complier. So, the program can be as large as this, but then what happens all these programs by demand paging has to get executed only for the only using these 5 page frames. So, at most 5 pages from this will be in the memory and then there will be some similarly. So, what happens is the page directory of this will map these this OS page on to this ram, and the remaining pages whatever here will be basically mapped on whatever I am using here will be mapped onto these 5 page frames. So, there will be at 5 valid entries for these pages, the remaining whatever pages are there for this process this process is using those pages will be just invalid. And as when I keep moving from pages from here to here and going it back, back and forth the corresponding entries with the page table will be changed right.

So, as far as the complier is seeing it sees this their address space, that as far as actual execution it will have only 5 pages plus the OS page. So, the entry for the page table corresponding to P 1 will have entries for this OS plus will entries only for this 5 frames right and. So, the physical address will be restricted to these 5 frames, and where that physical address will say will depend upon which of those 5 pages are good right there can be 5 pages here loaded the corresponding entries will 0.2 these 5 frames maximum 5. When this process P 2 starts executing then again the OS page will be loaded and then there can be at most 7 pages from this lot. Say let us 100 pages, there will only 7 pages actually loaded into this maximum right and the remaining 93 pages will be invalid. Which of those 7 pages are loaded that depends upon your execution; are you able to follow? And there will be no more entry in this page table other than these 7 page frames. So, it all this process has to execute that whichever page is necessary has to be loaded into one of these 7 frames and execute it.

I cannot go and look at some other page frame other than these 7 frames, when this process P 1 is executing it can only execute on using these 5 page frames on that time it cannot go and use anything by this I am getting you and isolation between these two processes. So, when I am loading the page table itself only these pages are valid these page frames are valid. So, I cannot go and use any other page frame other than this and this is what is your physical memory. So, I actually get a complete isolation between P 1 and P 2 are you able to appreciate this? So, how is protection inter process protection ensured by paging this is how it is ensure right.

Now, the biggest drawback between you know the segmentation and this is that is only protection between the process is, but with, but in the same process itself intra process protection sort of things cannot be ensured here right. Because now I cannot I do not have a way of specifying saying that this is a code segment, I do not have a way of specifying that this is stack segment, I do not having way of saying this is the data segment I say it is a page right, but I can certainly say that P 1's frames and P 2's frames I have isolate. So, when P 2 is starts executing it will load some of these frames from the memory to the from the hard disc to the main memory. Now when P 2 finishes or let us say there is a round robin scheduling P 1 executes for some time then P 2 and vice versa.

When I when P 2 is executing it will bring some frames into this it will bring some pages into this frames. When it shifts to P 1 then what will happen? P 1 will also bring some frames here again when I shift back to P 2, I will re use those frames here are you getting these points. So, I every time so; that means, when P 1 is executing actually p twos instructions are still in the ram, but P 1 will not be allowed to touch those things because of my paging mechanism and again when P 2 is executing P 1's frame that it has brought it is still in the ram, but P 2 will not be allowed to touch it because of this paging and virtually P 1 and P 2 will be given 4 GB address space. Are you able to follow this? That is why when you do the task switching in your next assignment you will find that CR3 the context of a process will also have this CR3 register; that means, I could have a page directory for each process and since a page directory is only 4 kb and then I will have at most one or two page tables. So, with 3 pages I can main maintain the complete virtual memory per process. So, this per process paging is what is very very important are you able to get this ok.

Now going forward how did I decide? So, what is a immediate question that will come in your mind how will I decide that P 1 needs a 5 frames, P 2 needs 7 frames how will you guess this. Seriously there is no answer for this question of because if I say I have a very large program right it does not mean that it is going to take a large execution time right. So, if I have very short program does not mean it will quickly finish off can you give me one example of a very short shortest program that you have seen, which will take a normal amount of time.

Student: (Refer Time: 10:57).

Infinite, but let us say. So, complete a program that will eventually complete.

Student: (Refer Time: 11:04).

While one is something that there, but that nobody will let us talk of a sensible program, but it will take shortest program, but it will take lot amount of time I think yes one hundred thought you.

Student: (Refer Time: 11:26).

Right you cannot finish up in linear time.

The shortest program that will take large largest amount of time probably great grandchild will finish you see them that type of, but say seventy eight line program right.

Student: (Refer Time: 11:47), prime factorization

Prime factorization you will do that have written in your line towers of antenna I right I will I will done towers of antenna right it will take 2 power n 10 n disc moving and from one tower to another tower such that I small this should not be on top of the large larger, larger radius disc should not be on top of the small radius. So, I will have some n discs in one thing sorry and then I have 3, I have to move each 4 discs to this I can use one more rod here for the line ye do not say that you do not know towers of antenna do you know or not yes right. So, this takes 2 power n 10. So, they have started by the end time you finish this Kaliyuga will finish or something some stories also like this. So, that is you cannot say just by looking at the size of the program, you cannot say what is the memory that I need to allocate for it right.

So, I could have a very large program, but it will be just one straight line code, I could have a very small program that will be running across. A supposed P 1 has a 90 percent of P 1 will execute on 6 pages, then if I have just 5 frames then I am gone I need at least 6 pages there. So, then my thing will work. Please note that a page fault is a very costly affair why it is a very costly affair? Because I have and I land of the dirty page worst case I have to call I have to find out which page to replace, I have to copy that page back and then copy this back. So, a page fault will kill performance. So, we need to see that if I allocate the frames then my page fault should reduce you all understand what a page fault this right. So, the objective here is to reduce the page fault. So, I have to allocate that many frames so, that my the page fault actually gets reduced right.

So, there are two ways of looking at it operating systems course will teach you this when they do memory management in operating system course, they will teach you this in great detail. There are two types of allocation one is call static page allocation or page frame allocation should be call static frame another is an obviously, dynamic page frame allocation. What is static page frame allocation? When the program is about to become a process when the program starts execute at that point of time I say these are all the pages these many pages page frames are allocated to you. I will give it 3 page frames at the time of start. I will give 3 page frames at the time of start then what will happen that 3 page frames will be there for (Refer Time: 15:04) I do not dynamically change the number of page frames right, but there is another thing called dynamic page frame allocation where I see the program behavior and I keep allocating more pages. As the program is executing I make a decision of how many page frames I need to give to it right.

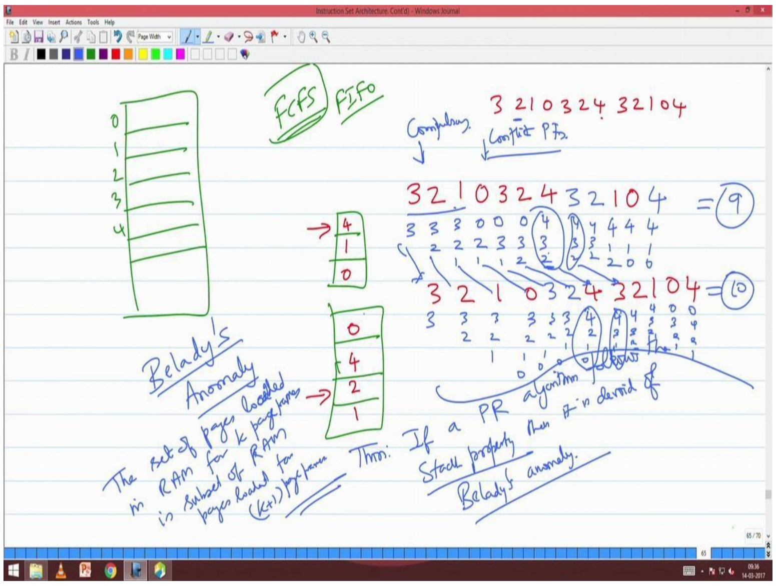
So, if the number of page frames allocated to a process is this is detriment before execution of that process then it is called static page frame allocation. When the number of page frames necessary is decided at while I am executing the process, then I can put machine intelligence there see what is happening I could intelligently keep increasing the number of page frames decreasing it etcetera right then it is actually called dynamic page frame allocation. So, what are different static page frame allocation techniques, what are different dynamic page frame allocation techniques this will be thought you in great detail in your fifth semester operating system memory management part of your course. But as far as this is concerned yes I allocate some frames and I use those frame and that is very important from an architecture perspective because, I cannot say that entire ram will be erased and reloaded while and when I switch from one process to another because I do lot of work as process P 1 or actually as process P 2, I spend lot of time to bring 7 frames here.

The moment that is a contests which is 7 frames in that ram cannot go off it has to be there. So, that I get performance otherwise every time when I go to a contest switch I go and fetch the entire t thing from you know virtual memory to ram it is impossible, then your performance will be even much slower than a turning machine right. So, you have to be extremely careful in doing this, are you are you able to follow what I am trying to convey here.

So, page frame. So, as a process comes into some other a process comes into then it has its own page table every process has its own page table, and the virtual memory of that process is basically filled up with some amount for OS, what are the things inside that OS you are already seeing you have seeing some GDT, LDTs. Now this page table then you will see something more like process block etcetera we will see as we go on to the next assignment for tasks witching. So, these are the things that will be there on the OS part. So, for the every fellow has to have this OS right and then the entire remaining space is dedicated to in the virtual address is dedicated to the process, but then on the physical side this entire process will be allocated some constant number of page frames which it will use to shift in and shift out pages to complete the execution. So, this is the entire setup and objective for the page replacement and page allocation. So, there are how many pages should I give you and what is the replacement strategy I need to follow these two things are taken care of by the operating system.

So, the next interesting thing is the question how do I the objective of this entire exercise is that I need to go and reduce the number of page faults. If there are more number of page faults then my performance is going to decrease tremendously. Now let us look at this now the way I can reduce page fault is to give more page frames correct so; obviously, we have.

(Refer Slide Time: 19:04)


Instruction Set Architecture Cont'd) - Windows Journal
FCFS
FIFO
Compulsory
Conflict PFs.
3 2 1 0 3 2 4 3 2 1 0 4
= 9
= 10
Belady's Anomaly
The set of pages loaded in RAM for k page frames is subset of RAM pages loaded for (k+1) page frames
Thm: If a PR algorithm follows the Stack property then it is devoid of Belady's anomaly.

Now, let us say that there is a program which is what are the maximum value 3 2 1 0.

Student: (Refer Time: 19:15).

4. So, it will using say 5 pages there is a program which is using 5 pages. So, let us say 0 1 2 3 4 5 pages is there and am giving it only 3 pages in the memory, the number of age frames I am giving is only 3 and the replacement strategy that I am following is the first come first serve trail which first will go on more right. So, the first in first out rather than first come first serve its called first came first out ok.

Now, let us say I am giving only 3 pages. So, initially these 3 pages initially everything is empty. So, the first page, what is that number 3 2.

Student: 3 2 1 0 (Refer Time: 20:07).

3 2 1 0.

Student: 3 2 1 0.

Ah.

Student: 3 2 1 0 4.

3 2 1 0 4. So, this is the way the pages are accessed first a program will start accessing page number 3 is starts main is there some routines or complied above. So, it is done. So, nothing is here. So, 3 is page fault. So, 3 gets loaded here now this is the start. Next two comes 2 is a page fault, next one comes one is also a page fault. Now 0 comes who gets replaced 3 gets replaced. So, this is now 3 comes now who gets replaced. So, this all this has been page faults. Now again two comes who gets replaced, one gets replaced right now 4 comes who gets replaced 0 gets replaced right. Now 3 and 2 that is not have any page faults 3 is there 2 is there already again one comes now one comes this fellow gets replaced now 0 comes then 4 comes there is no 4 is already there in it.

So, how many page faults it cost? 9 page faults it has cost right. So, the program what t this is this is called this is the page sequence page access sequence the program started executing from page 3 is starts it execute a lot of instruction page 3, then it started accessing page 2 it is not that this instruction is page 3 next instruction is page two what do you see by this sequences page 3 fetch and I will execute lot of things there then go to page two assemble point and then. So, this is the page access sequence this is not it does not means that instruction 1 is in page 3 instruction 2 is in page 2 you got it I bring page 3 execute it something then page 2; so then.

So, totally 9 faults where there 9 page faults happened; in which please note that the first 3 2 1 etcetera they were all compulsory faults they have called compulsory faults because they have to be loaded. The remaining thing got where basically because we loaded something and now we undo it and then do something. So, after 3 2 1 came 0 got pre loaded right. So, this is called compulsory faults page faults and the remaining called conflict page faults they came because of conflicts this 0 came in because 0 was not there in that page, but it was not that there was and empty space everything was filled and. So, we went and replace something and put 0 we replace something with 0. So, that is why we call it as conflict page faults.

So, totally 9 page faults happened. So, I am not very happy with this 9 I need to go and reduce this. So, what is the next way of reducing? It give it 4 now what happened then next 3 and 2, and the next 3 and 2 hour all page history and tour next is 4 what happens 3 is a page fault 3 2 1 0 are page faults then next 3 and 2 and the next 3 and 2 are all page hits because 3 and two are here the next is 4. So, what will happen to 4? 4 will get loaded here. So, 4 will get loaded here the next was 3. So, 3 will get loaded here the next was 2, the next was 1, the next was 0, the next was 4.

So, the number of page faults now is 3 1 2 3 4 5 6 7 8 9 10 by increasing the number of page frames I mean number of faults actually increased it is not decreased correct right. So, earlier case it was just 9 now it is 10 right this anomaly even me could have found out, but well before our birth Belady found it. So, this is called Belady's anomaly. So, by the end of your thing you also find out some anomaly will say. So, we can put your name there now this is this is called Belady's anomaly.

What do you mean by Belady's anomaly what is the consequence of Belady's anomaly? Just by increasing the number of page frames I cannot go and decrease the number of page faults right. So, the way a program executes is not you know capturable by some method. Now what how can I rectify Belady's anomaly. So, there is a theorem which is very straight forward to prove if a page replacement I will call it as PR, if a page replacement algorithm follows the stack property then it is divide of Belady's anomaly if a PR algorithm follows the stack property right then it is devoid of Belady's anomaly.

What is the stack property the set of pages loaded in ram for k page frames is subset of ram pages loaded for k plus one page frames. So, I use this algorithm for k page k given page frames at every instance there will be some set of pages loaded. Now I increase the number of page frames to k plus 1 and again run the algorithm again run the program at every instance there will be some set of pages loaded right. Now the set of pages loaded for k page frame should be a subset of the set of pages loaded for the k plus one frames then Belady's anomaly will not have why Belady's anomaly will not happen if Belady's anomaly happens how do you argue this through let us come to the proof letter, but let me show here. So, what has happened initially 3 alone was there then 3 2 was there the 3 2 1 then 0 2 1 0 3 1 then 0 3 2 then 4 3 2, 4 3 2, 4 3 2 then 4 sorry 4 1 2 then 4 1 0 and then this is again 4 1 0.

Now, let us do this 3 3 2, 3 2 1, 3 2 1 0. So, this is 3 2 1 0, 3 2 1 0, 4 2 1 0, 4 3 1, 4 2 1 4 3 1 0, 4 3 2 0, 4 3 2 1, 0 3 2 1, 0 4 2 1. Now you see 3 and 3 there is no issue 3 two there is no issue 3 2 1 there is no issue 3 2 1 0 there is no issue 3 2 there is no issue 3 2 there is no, but when I take this 4 3 2 right this is 4 2 1 0, but that is 4 3 2. So, this is not a subset of this right and let us take this 4 3 2 this is 4 3 1 0. So, this is not a subset of this is not a subset of this that corresponding location again here 4 3 2 right. So, this are the reasons why you know this whole thing goes into here are you able to follow right.

So, essentially what has happened the stack properties violated by the FcFs. Fcs does not ensure that the set of pages loaded in ram for k page frames is a subset of set of pages loaded for k plus 1 page frames and that is prissily the reason why while we land up with this Belady's anomaly are you able to follow.

Now, can you go and prove Belady's anomaly can you go and prove this statement that if a page replacement algorithm follows the stack property; that means, set of pages loaded in ram for k page frames is subset of ram pages loaded for k plus 1, page frame if this satisfies then it is devoid of Belady's anomaly can we prove this what is the way we can do what is the shortest way of doing this.

Student: (Refer Time: 33:18).

That was not no, but page fault is, but I want to say that it will not increase if the stack property is satisfied then if I increase my page frame in page frames allocated to your process then the number of page fault will eventually has to decrees, but in this case it has increased how can you go about this prof. So, give me a prof a very quick proof.

Student: Contradiction.

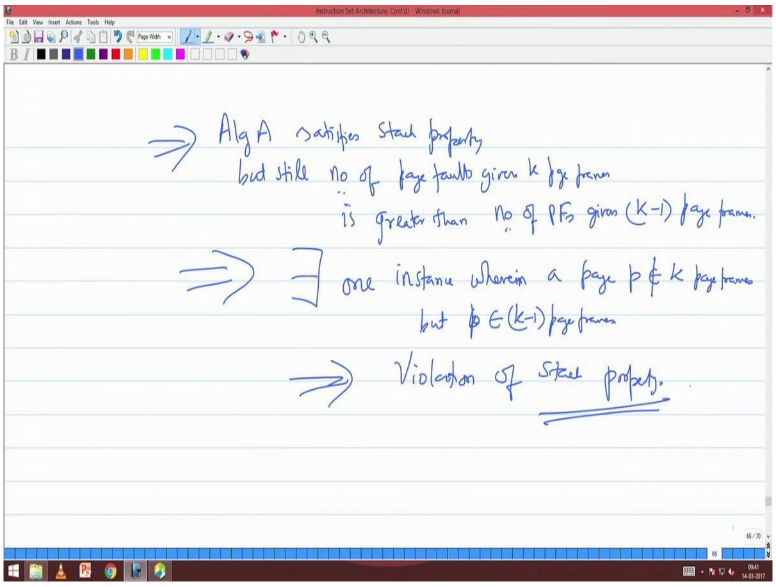
Prof by contradiction somebody said now what will you contradict and what will you prove prof by contradiction is correct.

Student: Suppose (Refer Time: 34:04) it increases when.

What do you.

Student: Number of page faults increased suppose.

(Refer Slide Time: 34:16)


$\Rightarrow$ Alg A satisfies stack property
but still no of page faults given k page frames
is greater than no of PFs given $(k-1)$ page frames.

$\Rightarrow$ $\exists$ one instance wherein a page $p \notin k$ page frames
but $p \in (k-1)$ page frames

$\Rightarrow$ Violation of stack property.

So, I have algorithm a satisfies stack property, but still right number of page faults given k page frames still number of page frames given k page frames is greater than number of page faults given.

Student: K minus 1.

K minus one page frames correct; that means. So, this is this is the contradiction algorithm a satisfies stack property, but still number of page faults given k page frames is greater than number of page faults was given k minus 1 page frames; that means, this implies they are exists at least one instance wherein.

Student: (Refer Time: 35:37).

A page p belongs to k page frames K page p does not belong to k page frames, but p belongs to K minus 1 page frames right this implies right because there should be some instance where k minus 1 does not suffer from a page fault and k suffers from the page fault the instance of k page frames suffers then only I can get this more right. So, this implies this violation of stack property. Hence if my algorithm satisfies stack property then I say keep increasing the number of page frames my page fault will decrees unfortunately in the case of FcFs since stack properties violated Belady’s anomaly can basically keep in it are you able to follow right.

So, what will do is in the afternoon class I will come I will talk about a little more about paging and these are all now this and the next chapter will be more towards the operating system, because as you are see we are slowly moving towards operating system concepts. So, we will discuss more in the afternoon class.

Thanks.