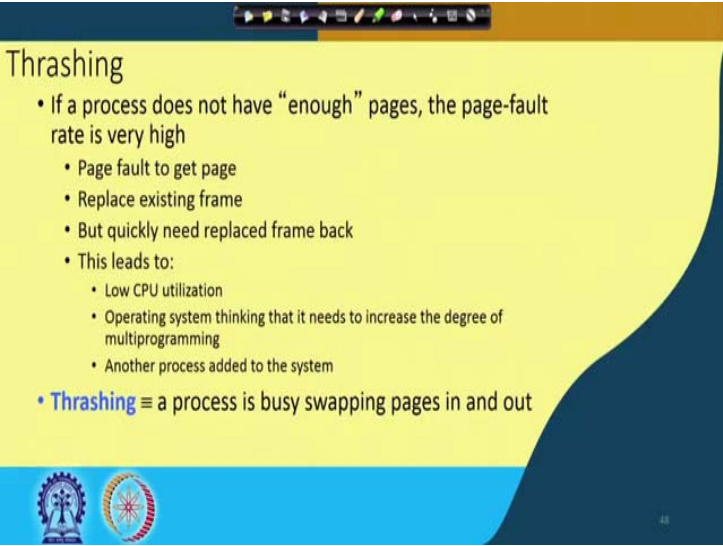


Operating System Fundamentals
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 55
Virtual Memory (Contd.)

In our last class, we have been discussing on the problem of thrashing.

(Refer Slide Time: 00:30)



The slide is titled "Thrashing" and features a yellow background with a dark blue curved shape on the right side. At the bottom, there is a blue bar containing two logos: the Indian Institute of Technology Kharagpur logo and a circular logo with a gear and a sun. The text on the slide is as follows:

Thrashing

- If a process does not have “enough” pages, the page-fault rate is very high
 - Page fault to get page
 - Replace existing frame
 - But quickly need replaced frame back
 - This leads to:
 - Low CPU utilization
 - Operating system thinking that it needs to increase the degree of multiprogramming
 - Another process added to the system
- **Thrashing** \equiv a process is busy swapping pages in and out

Now, as we can understand that you should have enough number of pages for every process in the memory so, that this page fault rate is low. But what happens is that if we want if we want to allocate enough pages to each enough pages to each process or enough frames to each process. Then the number of frames or the amount of main memory that will be needed will be huge; considering that number of process is large. So, it will require a huge amount of memory.

So, naturally we have to restrict the number of frames located two processes. Now, as a result, there may not be enough pages for a for a process. So, if a process does not have enough pages, page fault rate will be high, because every now and then some page will be required, which is not there in the memory.

And, this is particularly true when you have got local replacement sort of problem like you, that the same page is the same page may be swapped out again and again. And, the

page whatever is swapped in is again swapped out because of the necessity of new pages to be loaded.

Now, if this happens then the page fault is going to be high page fault rate is going to be high. When page fault occurs to get a new page from the secondary storage to the main memory so it may require replacing the existing frame, because the memory frame may not be free. So as a result it has to select a victim page and that victim page has to be removed from the frame and the frame made available for the new page to be loaded.

But, since that page that you have replaced it may be needed again very soon so, as a result the quickly we need the replaced frame back. So replace page back. So, as a result this page fault rate will increase. So most of the time the system will be busy doing this swapping in and swapping out of the memory frames the pages in the memory frame.

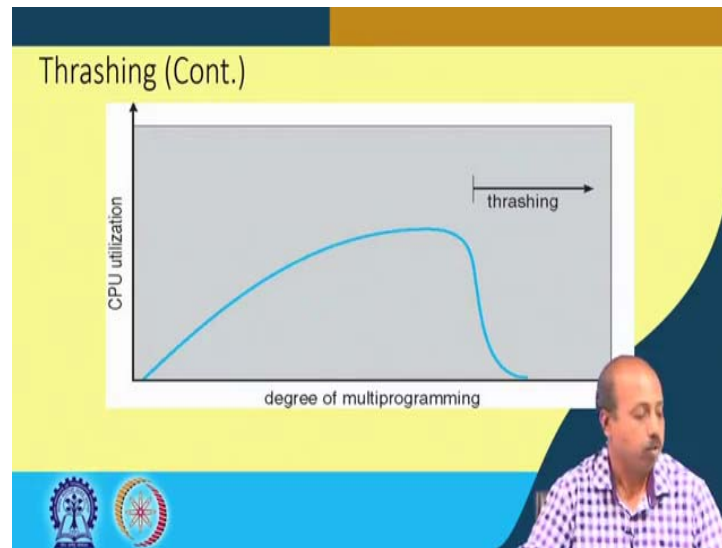
So, as a result the CPU utilization there is number of jobs completed per unit time. So, that value will become low. And, as that value becomes low, as we have seen previously in the while discussing on CPU scheduling issues that one possible reason why that the CPU utilization maybe low is that, the degree of multiprogramming. That is number of processes in the system or number of processes in the ready queue that may be low.

So that may instruct that long term scheduler to introduce more processes into the system. So more number of processes will make transition from new to ready state, but for that again we need to allocate memory frames to those new processes that have been admitted. So, that again increases the possibility of thrashing.

So, this process continue so, as a result a very you know very significant degradation may occur in the system performance, in terms of number of processes computed per unit time.

So, thrashing is equivalent to a process during busy swapping of pages in and out.

(Refer Slide Time: 03:33)



So in fact in this graph you can understand that as we are increasing the degree of multiprogramming. So initially when the degree of multiprogramming is low so, there are very few jobs completed per unit time. So, this the CPU utilization is also low. As you are pushing in more and more jobs into the system so, degree of multiprogramming will improve, but after a certain level since the number of frames allocated to processes is limited. So it will come to a stage where the thrashing will start. And, when this thrashing initiates, then the system CPU utilization is start dropping again.

Now, definitely it will not touch 0, because after all it is a CPU is doing something. So, it these access will not touch 0, but the CPU utilization will drop significant. So, we are bothered about this situation when this thrashing initiates. So somehow we have to stop this thrashing. So, somehow we have to do something do some strategy. So that this thrashing is restricted. At the same time we do not want to allocate large number of frames to the processes, because then this memory utilization will again below.

(Refer Slide Time: 04:44)

Demand Paging and Thrashing

- Why does demand paging work?
 - Locality model**
 - Process migrates from one locality to another
 - Localities may overlap
- Why does thrashing occur?

Σ size of locality > total memory size

- Can limit the effects of thrashing by using:
 - Local page replacement
 - Priority page replacement – replace a page from a process with the lowest priority.

The slide includes a diagram on the right showing a vertical stack of memory frames. Some frames contain horizontal lines representing pages. A circle is drawn around a specific page in one of the frames, and an arrow points to it from the text 'Process migrates from one locality to another'. Below the diagram is a small number '2'. At the bottom of the slide, there is a Windows taskbar with various icons and a video feed of a man in a checkered shirt.

So, how to address this problem that we are going to discuss today? So, demand paging and thrashing. So, why does demand paging work? This works because of the locality model, because demand paging says that you initially load only a few pages of a of a process. And as and when the process needs more pages so, it refers to addresses which are not there in those pages, then it the required pages will be loaded.

Now, due to the locality of programs as we have discussed previously so, this process it refers to a certain part of it is code or certain sections of the big data structure. So, that is why this locality happens and this demand paging policy works. As the process migrates from one locality to another so, we have got this.

So, as a process as a process migrates from one region to the other; so, if you can if you look into this process structure may be the this is the these are the pages of the process. So, initially the process may start in say in these two these two pages are loaded. So, initially the main program is starting here. So, it will be consulting address in this range only.

But, after that after sometime it will it will be referring to some other memory frames other pages. So, maybe now it will refers to say this page. And, this page is not being required for the time being. So, that is how the locality changes. So, if you consider this sequential lines of programs, then initially it execute first few lines, then after sometime when a procedure is called. So, it is going to that procedure and it is executing lines

within the procedures. So, at that time this program lines that were there. So, they are not needed.

So, this is how this locality changes as a process moves from one section of code to another section, the locality changes, the pages that are required changes. And, why does thrashing occur? So, thrashing occurs because the size of this locality is better than total memory size. So, if the locality of a process at some point of time if the locality says that it requires a this page, this page and this page. So, three pages are needed.

The reference pattern is such that very frequently this two pages are being referred to. So, called locality consist of three pages. Now, if I locate only two frames for the process, then what will happen maybe I be able to load so, this two pages at some point of time. So, natural there will be page fault for the third page. Now, when this page fault occurs again to load this page, I have to replace one of the one of the two pages that I have already loaded. So, free that frame and load this page there.

So, again the thrashing will occurs. So when this size of this locality is greater than total memory size. So, you should read it has total memory size allocated to the process. So, that there is the thrashing. So, can we limit the effect of thrashing? So, we can limit it by doing a local page replacement. So, if you one policies that like, if many processes are there and we are following global replacement policies, and thrashing may affect multiple number of processes. Compare to that if you are doing local page replacement, then it will only affect one or a few purposes only.

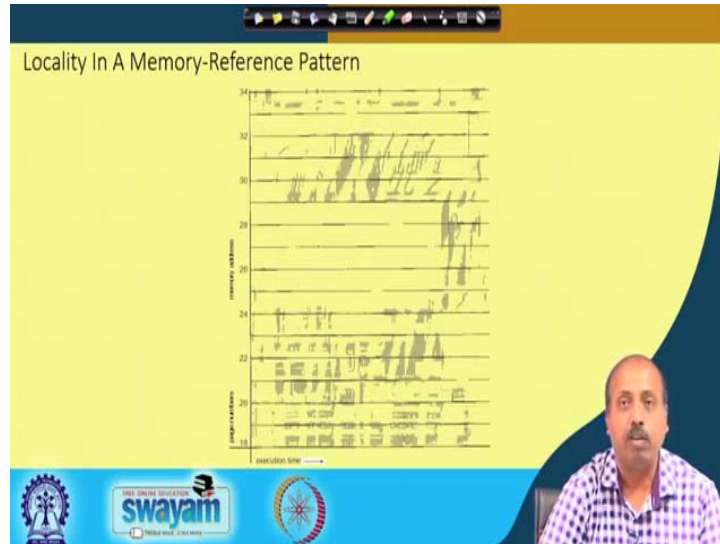
So, if there is single process for which this locality is not maintained properly, then thrashing will effect only that process in case of local page replacement. But, if we are following the global page replacement, then this miss behaving process or this process which is required requiring more number of frames in it is locality.

So, it will it will preempt other pages of other process. So, as a result global replacement will lead to more amount of thrashing. So, more number of jobs will be will remain incomplete for a longer time.

So, this local page replacement is one solution, another solution is the priority page replacement. So, we replace a page from a process with the lowest priority. So, that this

highest priority processes they do not suffer. So, these are the two possibilities that we can have you to solve this demand this thrashing problem.

(Refer Slide Time: 09:01)



Now, how to determine the locality and all that will try to see. So, you see that if this is a plot or this is a trace of this memory reference pattern. So, you so, if you see over the time over the timeline. So, initially this page numbers 18 to 20 so, they were referred at the beginning, but at that time this pages 26, 27, 28. So, they were not referred much.

At some point of time you see that in this region that is in this time zone ok so, in this time zone the pages 29 29 30 31 32. So, they are being referred to. So, if I can have this four pages loaded into memory at this during this interval of time, then there will be no page fault. Similarly, as you move forward so, if you come to say this region. So, in this region the pages 27, 28, 29 so, they are being referred to. So, that way they are being used.

So, in this way you can understand that this locality changes over time. So, this particular plot it is typical it is a typical program trace and it plots the memory address references over time as the program is executing. So, for any program you can generate these stress and if you plot it you can find that this referencing pattern over the time changes. So, at any point of time, if you know what is the differencing pattern and if you can ensure that the corresponding pages are there in the memory frames, then naturally this page fault rate will be low and this thrashing will be less.

And, as you are going from say one time zone to another time zone like, after say this time is over, now you see that this pages 31, 32; they are being referred to more. So, they initially they were not previously they were not there in the in the memory frames.

So, now, there will be page fault and the page number 32, 31 so, they will be loaded and again page fault rate will decrease. So, as a result as you are proceeding over times so, if you can keep a track of the locality of a process, then the you can load the corresponding pages into memory and then this page fault rate can be reduced.

(Refer Slide Time: 11:20)

Working-Set Model

- Define Δ to be a working-set window. Δ is a fixed number of page references For example: 10,000 instructions
- WSS_i (working set of Process P_i) is defined to be the total number of pages referenced in the most recent Δ (varies in time)
- Example with $\Delta = 10$

page reference table
 ... 2 6 1 5 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2 3 4 4 4 4 4 ...

Δ ← | t_1 Δ ← | t_2
 $WS(t_1) = \{1, 2, 5, 6, 7\}$ $WS(t_2) = \{3, 4\}$

So, this has lead to one interesting model which is known as working set model. So, we define in a quantity delta to be the working set window. So, this is the window of pages that are being referred to at in the near past and future of a time or time of a point in time. So, delta is a fixed number of page references for example, say 10 000 instruction. So, what 10 000 instructions, what else happened? The, what are the page references over the last 10 000 instructions.

Then, WSS_i is the working set for process P_i it is defined to be the total number of pages referenced in the most recent delta time. For example, with delta equal to 10 so, with delta equal to 10. So, you see that so, if you we are at time say; so, if you are looking at time t_1 . So, you look back 10 previous references like, say up to this much and you see that the pages that are referred to in this time zone.

So, they are 1 2 5 6 and 7 assuming that this process has got large number of pages. So, if you consider say this time zone, then the pages that are refer to our 1 2 5 6 7. So, had it been the case that this memory frames where already having this pages in there in the in the frame so, then the page fault would not occur.

Similarly, if you are looking at time t_2 , then over the last 10 references the pages that are referred 2 are 3 and 4. So, there is a significant departure from the previous working set to the new working set. So, WS working set at t_1 was 1 2 5 6 7, working set at t_2 is 3 and 4. So, as you are going from one region of code to another region of code or one part of data structure to another part of data structure this working set of the process will change.

So, ideally we have to catch this working set and then accordingly we have to ensure that the working set pages are there in the memory. And, they are not going to be replaced by the page replacement policy.

(Refer Slide Time: 13:35)

The slide is titled "Working-Set Model (Cont.)" and contains the following text:

- WSS_i -- tries to approximate the size of the locality of process P_i
 - if Δ too small will not encompass entire locality
 - if Δ too large will encompass several localities
 - if $\Delta = \infty \Rightarrow$ will encompass entire program
- $D = \sum WSS_i \equiv$ approximate the total demand frames
 - Approximation of ALL localities
- m = total number of frames.
- if $D > m \Rightarrow$ Thrashing
- Policy: if $D > m$, then suspend or swap out one of the processes

The slide also features a video inset of a man in a purple and white checkered shirt speaking, and logos of institutions at the bottom left.

So this is the working set model. So, WSS_i we try to approximate the size of the locality of process P_i . So, so this if delta is too small so, it will not encompass the entire locality like in previous example if I make so, with delta equal to say 2 or 3, then I will have only 1 7 5.

So, that will be missing the page 6 ok. Similarly, if I if I make the working set model only 2 so, it will have only 1 and 5. So, it will miss this 7 all together. So, that way if the delta if delta is too small, it will not encompass the entire locality.

If, delta is too large then it will encompass several localities like, if you say from time t_2 so, if you are looking back. So, if delta equal to say 100, then it will have all this pages 1 2 3 4 5 6 7 all the pages of the process, they will be in the locality.

So, that is of course, not desirable. So, you do not want that type of information. So, we want more localized information. So, naturally I cannot make delta to be too small, I cannot make delta to be too large. So, if delta is equal to infinity then naturally the entire program is in the locality. So, these are the extreme values of delta. So, choosing the value of delta is definitely a critical parameter and it is problem specific ok. So, like one program running so, it will have some delta value will be something else.

So, and also it depends on the environment in which you are running the program. So, depending upon say; if it is some payroll processing type of application, then this delta reference in pattern. And, if you are having say one whether forecasting application then this delta referencing pattern so, they will be different. So, all these are there. So, the it is not very easy to predict this delta, but we have to choose a proper delta. So, that is for sure.

Now, D so, define the quantity D to be some of these WSS i . So, for all the processes if we have computed their working set, working sets, then if you some them up, then this is the approximate total demand for frames. So, for all the all the process if we sum the working set, and that gives the number of frames that should be there in the memory under a demand paging environment. So, this is an approximation to all localities.

So, if m is the total number of frames and D happens to be greater than m so, at one point of time if I have admitted in number of processes, say P_1 to P_{10} and for each of them I know their sets at some point of time. Then you sum the working sets sizes and if the if the working set sizes are more than the number of frames that are available in the system, then definitely there with thrashing. So, this is the one indication that thrashing is going to occur if D is greater than m .

So, what is the policy? So, policy can be like this that if D becomes greater than m , in that case you suspend or swap out one of the processes. So, if you suspend or swap out then naturally that that processes frame requirements are removed so that will reduce this value of D and possibly it will come below m .

So at any point of time you cannot cross the value of m , because this thrashing will now thrashing will occur from that point onwards.

(Refer Slide Time: 17:59)

Keeping Track of the Working Set

- Keeping the exact information about each working set is impractical since the working set window is a moving window.
- Approximate with interval timer + a reference bit
- Example: $\Delta = 10,000$
 - Timer interrupts after every 5000 time units
 - Keep in memory 2 bits for each page.
 - View the reference bit and 2-in memory bits as a register (3-bits), with the reference bit being the most significant bit
 - Whenever a timer interrupts shift to the right by one place the 3-bits and set the values of all reference bits to 0.
 - If a page fault occurs, we can examine the 3-bits to determine whether a page was used within the last 10,000 to 15,000 references (at least one bit is on). If so the page is in working set
- Why is this not completely accurate?
- Improvement = 10 bits and interrupt every 1000 time units

So, how can I keep track of this working set? Working set once it is told that this is these are the working set for the process now that is fine, but how do I keep track of this working set, that is a very important problem. So, keeping the exact information about each working set is impractical, since the working set window is a moving window.

So, though in the previous example or previous formula so, we have said that D equal to $\sum WSS_i$ sum of WSS_i . But, what is the WSS_i ? So, WSS_i this working set or WS rather the working set of a process it changes as the process is executing. So, at maybe at time t_1 when I probe the system, I find that the process is referring to some procedure P_1 , at time t_2 when I refer to so, it is referring to some other setup procedure. So, as a result, the pages that it is referring to changes over time.

So the working set of the process will also change and this working set window is a moving window. So, approximate time approximate with interval timer plus a reference

bit. So, we do an approximation in which we keep a timer and a reference bit. So, if there is an example suppose I have taken delta equal to 10 000, that is that we are we will be considered in previous 10 000 references for computing the working set.

Timer it interrupts after every 5 000 time units and we keep in memory 2 bits for each page, view the reference bit and 2 in when 2 in memory bit in memory bit so, as a register so, total 3-bits. So, we have got we have got 3-bits. So, we have got 3-bits; one is the; this memory reference bit and these are the two extra bits, that we are talking about ok. So, this 2 bits of extra for that.

With the reference bit being the most significant bit so, reference bit is the most significant bit. So, when the timer interrupts we shift the shift to the right by one place, the 3 bits and set the values of all the all reference bits to 0. So, what we do, when the timer come. So, this is a shifted ok. And, then this reference bit is set to 0. And, reference bit will remember that, if that particular page is referred then that reference bit will be turned on. So, in the previous 5 000 time interval; so, if this page was referred to then this particular bit was one otherwise this bit was 0 it was off.

Now, when this timer comes then it is it is shifted right by 1 1 bit position. So, this reference bit comes to the next position. So, this reference bit is now shifted here. Now, so, if a page fault occurs, we can examine the 3-bits determine whether the page was used within the last 10 000 to 15 000 references at least 1 bit at least 1 bit at least one of these bits will be one. So, if the page is in whether the page is in working set or not.

So, what is happening is that. So, we are we are when these timer interrupt occurring. So, we are setting all the 3 bits to 0 0 0. So, if a page if a reference has occurred. So, this has turn to 1 ok. So, when the timer comes then we are shifting it by 1 bit. So, if where it is happening like this. So, it is it is now shifted by 1 bit position.

If this page is not referred in this before the next timer interrupt comes, then what will happen is this content will become 0 0 1. If, the page is still not referred when the next timer interrupt comes then this become 0 0 0. So; that means, you can understand that for previous so, many references. So, this is the 5, 000 plus 5, 000 10, 000 to 15, 000 to 15, 000 references. So, this page was not referred to if the content becomes all 0.

If, the content is not all 0, then what will happen is one possibility is that this reference bit 1. So, basically this situation, then definitely the page has been referred or maybe the situation has become 0 1 0 1 0. So, it was it is not it was not referred immediately, but it was referred in the previous (Refer Time: 21:16).

So, as a result this is 0 1 0 or see 0 0 1, that also means that it was it is not referred immediately, but to 5 000 step backward. So, it must have been referred to. So, if a page fault occurs, then we can try to select a page from this one. So, this page is if all the bits are 0, then that page is definitely not in the working set. So, if a page fault occurs we can examine the 3 bits to determine whether the page was used within the last 10 000 to 15 000 references. So, if so, the page is in working set. So, if it was referred to then it is in the working set. So, we will try to replace pages which are not there in the working set of the process. So, why is this not completely accurate definitely? So, maybe that after 3-bit so; we are not keeping enough information. So, maybe there are two pages which are referred to like say one page was not referred for last say 20 000 cycles and the other page was not referred for last 15 000 cycles of both the cases, this bits will become 0 0 0, but that is not accurate.

So, the first one that I refer talked about 20 000 reference bit is not there. So, that is actually really outside the working set and this one maybe just it missed by one reference and that is why this has become 0 0 0, that is not a very correct estimation of the working set.

So, how can I improve this? So, improving the performance improving the scheme definitely I have to extend this number of bits. So, previously I just kept 2 bits. So, instead of that if you put a 10 bits for this additional memory bit to store the these references. So, instead of having only 2 bit so, you have got say 10 bits so, where we are studying the previous references.

And, instead of interrupting every 5 000 unit we interrupt every 1000 time units, that will give us more frequent information about this change in the working set. So, this is a model this is a strategy that can be followed for keeping track of this working set of a process and by using this reference bits.

So, we can shift this reference bits bit and then if the content is non-zero; that means, the page was referred to in the near future. So, it is in the working set. If, the content is 0;

that means, it is not referred to in the last 15 000 to 15 000 cycles whatever be the size of my delta based on that. So, it can be removed from the memory if required. So, this way we can compute the working set dynamically.

(Refer Slide Time: 24:02)

Page-Fault Frequency

- More direct approach than WSS
- Establish “acceptable” **page-fault frequency (PFF)** rate and use local replacement policy
 - If actual rate too low, process loses frame
 - If actual rate too high, process gains frame

Handwritten notes on the slide: $P_1 \rightarrow 4 \text{ frames}$, $W=3$, and a list of numbers 7, 8, 4, 5.

The graph shows 'page-fault rate' on the y-axis and 'number of frames' on the x-axis. A curve shows that as the number of frames increases, the page-fault rate decreases. Two horizontal lines are drawn: an 'upper bound' and a 'lower bound'. The area between these lines is labeled 'increase number of frames' and 'decrease number of frames' respectively.

Another, very important issue that we have is that page fault frequency. So, this is the more direct approach than WSS so, working set size. So, that was one consideration by which we can we can keep a track of this working sat of a process and that way, we can try to reduce the number of page faults. Another direct observation is what is the page fault frequency? The page fault frequency is very high; that means, the system must have gone into a thrashing situation.

So, this is the more direct approach than working shape size and we establish acceptable page fault frequency rate, and use local replacement policy. If, the actual rate is too low, then the process will lose frames and if the actual rate is too high and the process gains frames. So, it is like this.

So, if I have given say a process that comes suppose process P 1 has come. So, P 1 I have allocated say 5 frames. Now, depending upon the nature of the process, this 5 frames may or may not be sufficient for the process. So, if the if the number of pages the that the process needs is in in it is working set is say less than 5, suppose the P ones working set is equal to say V equal to say 3.

Then, at any point of time it will be referring to about 3 pages in if you considered successive references, then it will see that it is referring to 3 frames not the 5 frames. So, as a result the page fault frequency will be very low. So, there will be almost no page fault, and if that thing happens then we know that we have given extra frames to the process. So, can possibly cut down this from 5 to 4 and even after reducing to 4 if we find that the page fault rate is low so, we can even cut it down to 3. So, that way we can reduce the number of frame.

So, this is the thing that is actual rate is too low. So, we so, we agree upon some “acceptable” page-fault frequency for a system. And, if a process is found that it is requiring its page fault rate is much less than this PFF value; that means, the process does not need that much of frame. So, we can release some of the frames from the process.

So, if the actual rate is too low then process from the process we take back the frames and if the page fault rate is pretty high compare to this PFF. So, if the rate actual rate is too high, then we know that the process has perhaps not been able to keep it working set in the memory. So, we have to allocate it more frames.

So, that way we want to allocate it more number of frames p initially might be that maybe that we had allocated $P - 1$ only 2 frames. Now, this page fault frequency for the process is high. So, we increase it to 3 to 4 to 5. So, may be after coming to 5 I find that now page fault rate has decreased significant it is below the PFF rate. So, we can stop at that point, we can allocate 5 frames to the process.

So, this is again a dynamic situation. So, if you plot over time so, as you are increasing the number of frames and these page fault rate. So, when the page fault rate is pretty higher than the number of frames is low and we keep a bound like. So, this is the upper bound and this is the lower bound. So, if this page fault rate is above this value it is above this value above this one. So, in that case we want we have to increase the number of frame. So, that if the system was operating in this region. So, it will be coming down it will be working in this region.

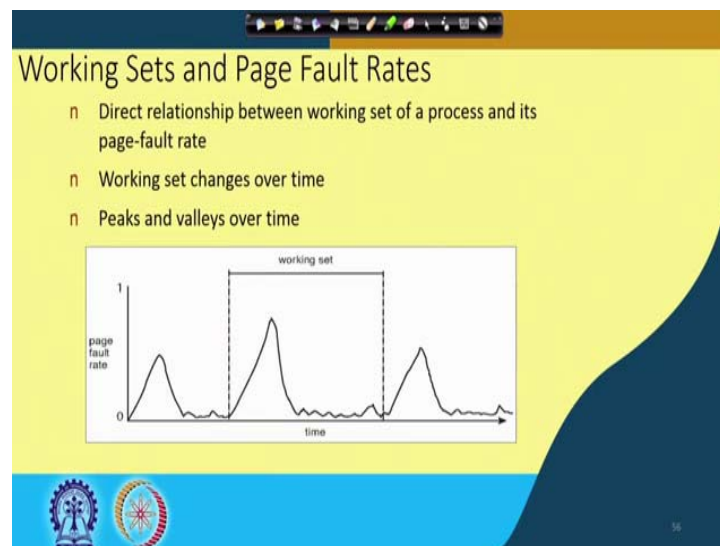
Similarly, if a process has been given large number of frames, then it will be operating in this region and then the page fault rate will be below the lower bound. And, if it is below the lower bound in that case we want to push it up. So, we want to take back the frames

allocated to the process we want to push it up and then the process will again come and operate in this region.

So, ideally the process should operate in this region of the car ok, should operate in this region of the car. And, if it is not happening then we have to adjust accordingly we have to see like, which process which whether the frame should be allocated more or less for the process, but this whole thing works with the local page replacement.

So, this is not for the global page replacement, unlike this WSS. So, which is a which can be for global replacement and it was controlling these degree of multiprogramming and all. But, here I do not have that so, this is for the local page replacement. So, this is not guiding your this long term scheduler to introduce more processes or remove more processes from the system.

(Refer Slide Time: 29:02)



So, then so, working set and page fault rate. So, if you see then you see over time. So, there is a relationship between this working set of a process and it is page fault rate. So, the page fault rate if we plot overtime, then maybe this page fault rate is going like this, when in this region this page fault rate is increasing means, it is the working set has not yet been loaded into the memory;. after this page fault rate decreases.

So, in this region of the of the graph so, we can infer that perhaps I have already loaded my working set in the memory so, as a result the page fault has reduced. Again you see

that in at this range so, again the page fault rate has started increasing. And again at this range it decreases and once this page, once it reaches the peak then the workings then workings then their page faults rate reduces. So, you can understand that in this region the page fault rate will be low because the working set has been loaded.

So, there is a direct relationship between working set of a process and it is page fault rate, working set changes over time and peaks and valleys overtime. So, this working set if you will plot page fault rate. So, you will get this peaks and values as. So, this way I can take help of this working set and page fault rate to tune the system. So, that this overall system performance becomes very well.

So, this way this demand paging has to work with this working set model and that can reduce the page fault and it is improve the system performance at the. So, that the degree of multiprogramming is also reasonable and CPU throughput is also high.