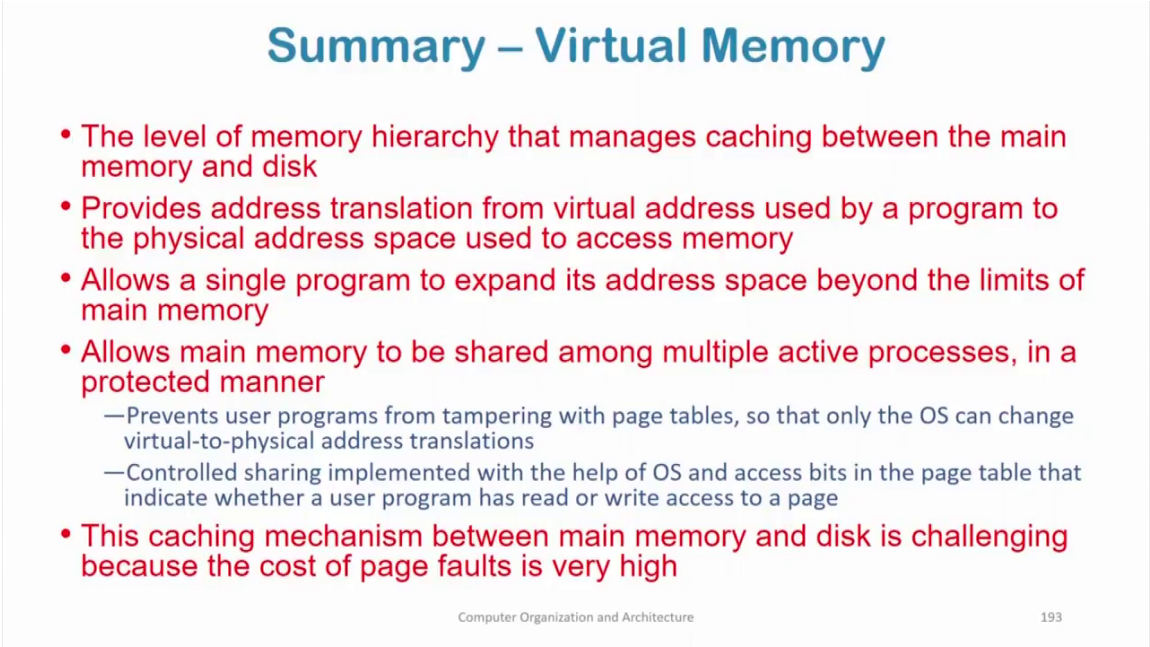
Computer Organization and Architecture: A Pedagogical Aspect

Prof. Jatindra Kr. Deka
Dr. Santosh Biswas
Dr. Arnab Sarkar
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture – 35
Summary of Memory Sub-system Organization

(Refer Slide Time: 00:26)



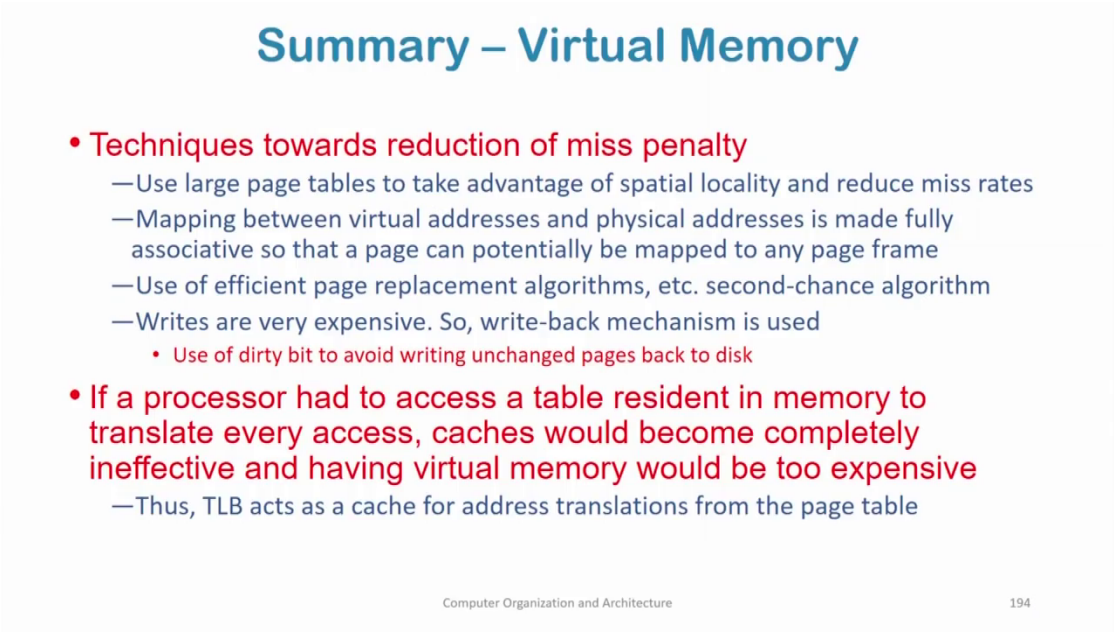
Here we will summarize our discussion with Virtual Memory. Virtual memory may be described as the level of memory hierarchy that manages caching between the main memory and disk. Therefore, it allows main memory to act as a cache for the disk. It provides; virtual memories provide address translation from virtual address, used by a program to the physical address space used to access memory. It allows a single program to expand it's address space beyond the limits of the main memory.

So, through this address translation because it allows the translation of virtual memory addresses to physical memory addresses through this scheme it allows a single program to expand it is address space beyond the limits of the main memory. It allows main memory to be shared among multiple active processes in a protected manner. How do you give this protection? It prevents this protection is given by preventing user programs from tampering with page tables so that only the OS can change virtual to physical

address translations. So, this is how you stop one program from accessing data of another program; although, multiple programs are sharing the same physical memory. However, it also allows controlled sharing of pages between different programs.

How? Controlled sharing is implemented with the help of OS and access bits in the page table that indicate whether a program has read or write access to a page, ok. So, if we can share a page between multiple programs through this access bits with the help of the OS. This virtual memory which we have described as the caching memory between the main memory and disk, this caching mechanism between main memory and disk is challenging because the cost of page faults is very high. If you have a miss in the main memory you have to go to physical memory. And we saw that this could be very high up to 100s of times slower, 1000s of times slower than accessing the main memory. So, the cost of page faults is very high.

(Refer Slide Time: 02:52)



So, we need techniques towards reducing the miss penalty. So, we don't want to go into the disk. So, we have to have techniques, we have to have techniques that reduce chances of going to the disk.

We use large page tables to take advantage of the spatial locality. Because misses in the main memory has a high penalty, we need to have techniques to reduce such miss

penalty. So, what are these techniques? We use large pages to take advantage of spatial locality and reduce miss rates. So, page sizes are of the order of 4 kb 8 kb or even larger.

Mapping between virtual addresses to physical addresses is made fully associative so that a page can potentially be mapped to any large into to any page frame. So, I can I should be able to put my page potentially into any page frame in main memory. A program can potentially put it's page into any page frame in main memory that is the main memory map this mapping is fully associative. And why it is does why does this fully associativity work here, because the misses the locality is higher and the misses are much more rare.

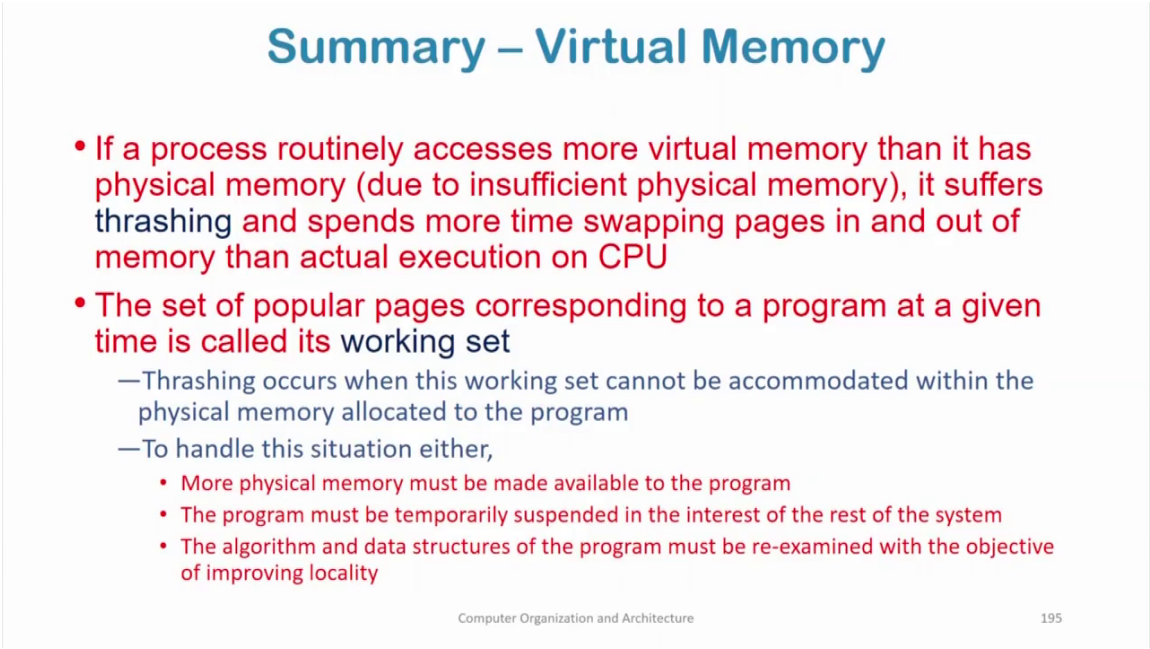
Use of efficient page replacement algorithms must be used, such as the second chance page page replacement which approximates LRU by using FIFO along with the reference bit. Writes into the disk are very expensive. So, we use a write back mechanism instead of write through. So, this virtual memories use a write back mechanism a page is replaced. So, when I am writing to ah when I am writing into the main memory I don't write into the disk. Only during replacement, I write back dirty page onto the disk, that is we I am we are using a write back scheme.

So, use of dirty bit to avoid writing unchanged pages back to the disk. Even within this suppose when we have selected, we saw that when we select a certain set of pages to be written to and we have written it back to memory. If that page is accessed ah if so when that page is written to the memory and it is free to be replaced. It is brought into the pool of free pages, even then we check whether the dirty bit of this page is on or off if the dirty bit is off, that page even though it is in the free frame pool, I can just use it because it is now unchanged. If a processor had to access a page table resident in memory to translate every axis, caches would become completely ineffective.

Now we said that page table is resident in memory. Now if I had to find out during or during an access when I am getting a page, if I had to go into the main memory to fetch the page table and get where from and get from where to get this page caches would become completely ineffective, because every access would ultimately require a memory access to access the page table. And therefore, use a virtual memory in that case would be very expensive.

That the TLB act acts as a cache for address translation from the page table. So, frequently accessed page table entries are therefore put in a TLB. Due to the use of the TLB to access the page table on every access, I don't have to go to the main memory right, and this improves the performance heavily.

(Refer Slide Time: 06:49)



If a process routinely accesses more virtual memory than it has physical memory due to insufficient physical memory it suffers thrashing as we saw. What is thrashing? in thrashing it spends more time swapping pages in and out of memory then actual execution on the CPU. The set of popular pages corresponding to a program at a given time is called it is working set. The pages the set of pages that that a particular program has accessed in the recent past is there in the working set. So, it is the set of popular pages corresponding to a program at a given time and it is called the working set. So, thrashing will occur when this working set cannot be accommodated within the physical memory allocated to the program.

To handle this situation, we can either have more we can allocate more physical memory, and to be to be made available to this process. So, I need to handle thrashing to reduce to reduce access, because I don't have all pages in the working set in main memory, how can I improve the situation? I will have to increase memory that is

allocated to this program. The program must be temporal or if I cannot do this this program must be temporarily suspended in the interest of the rest of the system.

So, I suspend this program, because this program is undergoing a lot of thrashing, and I will allow other processes to work smoothly. And it will be better, because the pages their page frames allocated to this process can be distributed among other processes in the process.

When the situation improves and the disk usage lowers disk access is lowered then this process can again be brought and executed. The other way to reduce to improve this situation of thrashing to reduce thrashing is to have an algorithm to have better algorithms and data structures for the program so that the locality of this program can be improved. When the locality of the program is improved, effectively the working set size of the program number of distinct sets distinct pages in the working set. So, the working set size therefore reduces. When this happens when the locality is improved, and hence the program will be able to work with a lower amount of physical memory.

So, these are the ways in which thrashing or thrashing can be reduced. With this we come to the end of this summarization on virtual memory.