Operating System Fundamentals

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

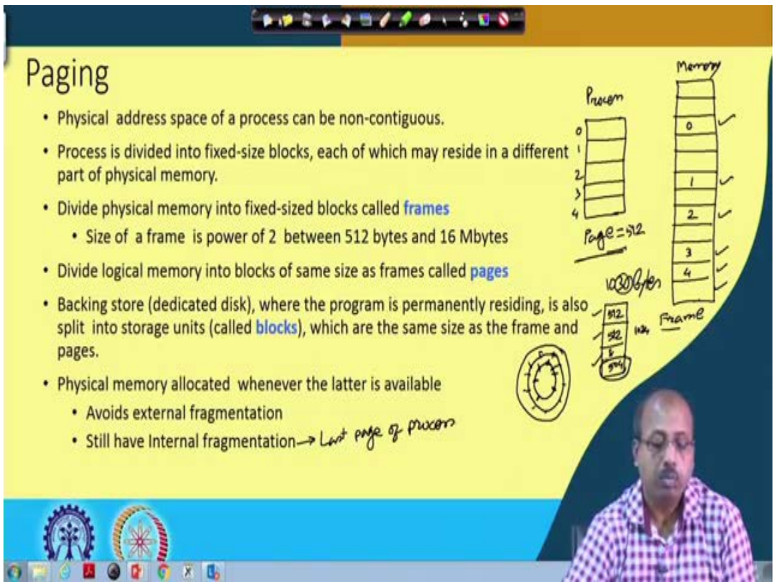
Lecture - 45
Memory Management (Contd.)

After looking into this segmented Memory Management, so where this process was divided into the modules developed by the user. Now as a programmer so, it is my prerogative to have this procedure structure the subroutine structures to be of different sizes, somebody may be having the habit of writing small procedures, somebody may be combining many of the small procedures together. Now what is the optimum size of the procedures so, that is a debatable issue.

So, we cannot expect that all programmers will have this segment this procedure sizes to be reasonable and as a result, this whole thing is done properly. So, as you can understand. So, if I have only a main program. So, there is no subroutine in my program, then what will happen? This entire piece of program becomes a single segment and memory management policy in that case any way turns out to be a standard that variable partition memory management.

So, that is no more a segmented memory management. I do not have any segments; I have got only one segment in my program that is the main program. So, as a result this entire exercise of this segmentation and all so, that is going to fail.

(Refer Slide Time: 01:49)



Paging

- Physical address space of a process can be non-contiguous.
- Process is divided into fixed-size blocks, each of which may reside in a different part of physical memory.
- Divide physical memory into fixed-sized blocks called frames
  - Size of a frame is power of 2 between 512 bytes and 16 Mbytes
- Divide logical memory into blocks of same size as frames called pages
- Backing store (dedicated disk), where the program is permanently residing, is also split into storage units (called blocks), which are the same size as the frame and pages.
- Physical memory allocated whenever the latter is available
  - Avoids external fragmentation
  - Still have Internal fragmentation → Last page of process

Now to avoid that situation so, next strategy that we are going to discuss is known as the paging mechanism. And in the paging mechanism, the operating system designer can have the flexibility so, operating system designer will divide the process into individual equal sized contiguous spaces and then that pages may be noncontiguous in the physical memory.

So, here what happens is that, this physical memory; so we have got two things one is the process that is developed by the user so, this process is say this is the process that we have ok. So it has got all these procedures, then these variables stack everything and this is the main memory that we have ok. So, what is done this process and this memory so, they are divided into equal sized say partitions. So, suppose this program is divided into 5 such partitions, which we call page. So, these are called pages. So, this is page 0, 1, 2, 3, 4. So, pages are of equal size.

Similarly, this memory the main memory that we have so, that is also divided into number of fixed sized partitions and that we call memory frame. Page size and frame size they are equal so, I can load any page into any frame. So, at any point of time, if you take a snap shot of the system, you may find that say this frames are the free frames, these are the free frames. Now while; so, when this process has to be loaded into memory, we find out the sufficient number of free frames. So, I need five such free frames so, maybe I load page 0 in this frame, then page 1 in this frame, page 2 in this frame, 3 in this frame, 4 in this frame.

So, that is also a noncontiguous allocation of memory space to the processes that is; so, the process pages are distributed in the entire memory space. So, the process is divided into fixed size blocks each of which may reside in a different part of physical memory and divide physical memory into fixed size blocks called frames. So, size of a frame is a power of two between 512 bytes and 16 megabytes.

So, they are made power of 2 so, that my addressing and this things becomes easier. And divide the logical memory into blocks of same size as frames called pages. So, this logical memory so, that is the memory of the process. So, that is divided into number of pages and this is the main memory is divided into number of frames.

And frame size is taken to be in different system, the frame size is different the page size is defined. So, it normally is a power of 2 going for 512 to 16 megabyte. Backing store which is the some dedicated disk, where the program is permanently residing is also split into storage units or blocks which are the same size as the frame and pages. So, what we have is that in the disk; that we in the disk we are storing the file ok. So, this process that is there so, corresponding process is also stored in the disk and in this disk is also divided into this will see later, that this disk is organized as some sectors finally.

So, this sectors that we have so, this small portions of this circular tracks that we have here. So, they are actually called; they are called sectors and each sector contains one block ok. So, this block size or the sector size is made equal to the page size or frame size. So, this sector size, page size and frame size they are all same. So, we can transfer one sector of data from this disk to the frame or it is also the page actually. So, this backing store, where the program is permanently residing is also split into storage units called blocks.

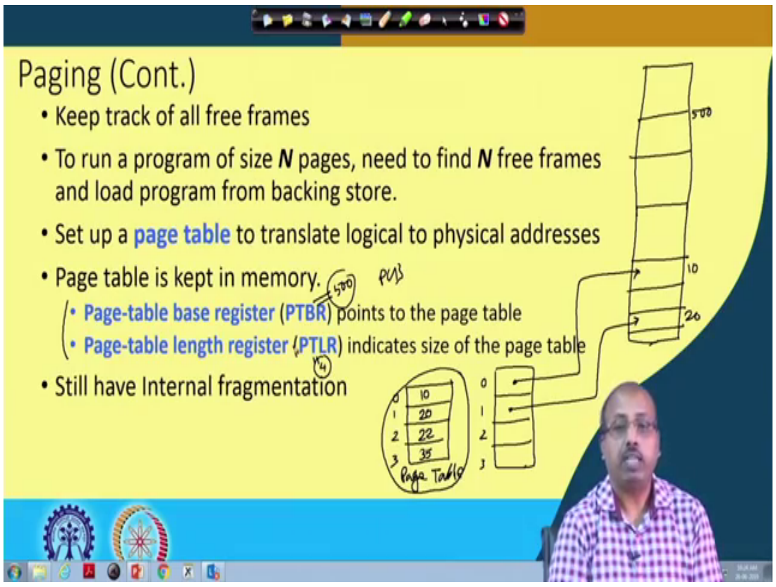
So, instead of telling sector we will call them block because instead of disk so, we can use some other type of storage also. So, block is a better term for that and that block size is same as the frame size and page size. Physical memory allocated, whenever the latter is available. So, avoids; so, whenever physical memory is any process has to be loaded so, you know that this process requires ten pages. So, you see whether ten pages are free in the, ten frames are free in the memory or not, if ten frames are free then we go for allocation. So, the advantage that we get is that the, it avoids external fragmentation. So, external fragmentation is not there because any frame that is free so, it can accommodate one particular page.

So, we are overhead is in terms of page and page size being sufficiently high so, we can always maintain the list of free pages. So, that is what storing the information. So, this external fragmentation problem does not occur; however, it can have some amount of internal fragmentation, because if I keep that page size to be equal to say 512 bytes, now a program may be equal to 1030 bytes. So, when I am trying to allocate it space so, this is dividing into frames. So, the first divide into pages; so, first page has got 512 bytes, second page will also have 512 bytes.

Now, what about the so, that makes us that takes into 1024 byte, but still I need 8 more bytes. So, 8 more bytes will come in the last page. So, this is only 8 bytes are used. So, remaining so, these 504 bytes that are there so, that is a wastage. But this 504 bytes I cannot this space I cannot utilize because, when I am trying to load this pages into memory frame so, they; this three will be loaded at three different frames and for the last frame so, this amount of space is going to be wasted ok. So, this I cannot utilize. So, that way internal fragmentation remains, the for the last page internal fragmentation this comes for the last page of the process. So, that cannot be avoided.

But any way; so, but external fragmentation is not there and this internal fragmentation happens for only for a single page of a process. So, that is tolerable. So, that is why so, this is going to be useful. So, will see that what sort of hardware support that may be needed for handling this particular type of memory management policy.

(Refer Slide Time: 09:19)

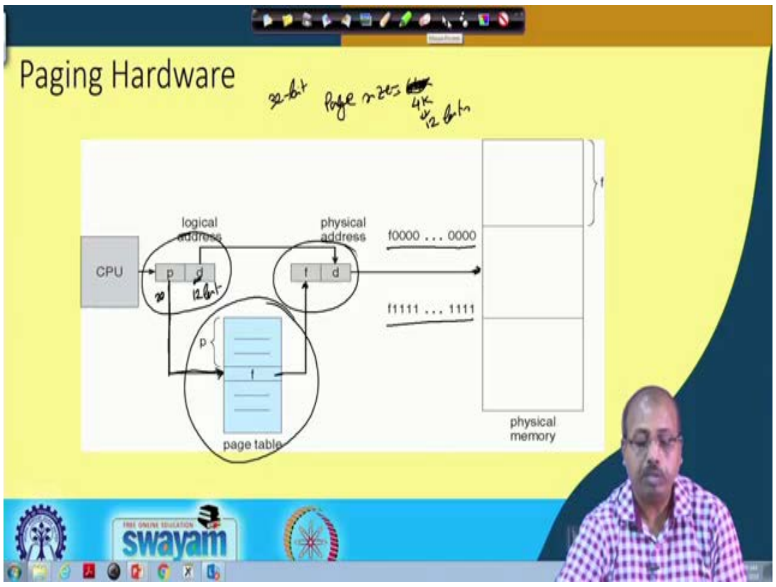


Paging (Cont.)

- Keep track of all free frames
- To run a program of size ***N*** pages, need to find ***N*** free frames and load program from backing store.
- Set up a page table to translate logical to physical addresses
- Page table is kept in memory.
  - Page-table base register (PTBR) points to the page table
  - Page-table length register (PTLR) indicates size of the page table
- Still have Internal fragmentation

So, keep track of all free frames to run a program of size N pages we need to find N free frames and load program from the backing store from the disk we need to store load the blocks into the free frames. And then we set up a page table to translate logical to physical addresses. So, will see some diagram then will come to this ok.

(Refer Slide Time: 09:44)



Paging Hardware

So, this is basically the C P U. So, C P U generates the logical address and logical address is divided into page and frame ok. So, for example, if the logical address if the C P U generated address is say 32 bit; the C P U generated address is 32 bit and this page size is equal to say 65 K aach page is of 64 sorry instead of say each page is of say 4 K. So, if each page is 4 K; that means, this requires 12 bits, this requires twelve bits. So, this displacement part so, this is the displacement within the page. So, this is 12 bits and this., then this remaining 20 pay bits so, they constitute the page part.

So, this page table will have so, this page number will be coming from this page part. So, that will be used to index into this page table; so, page table will give me the corresponding frame number. So, for a particular page in which memory frame it has been loaded so, that frame number will be coming from here and this frame number and this displacement they will be combined together get the physical address. So, that way it will go to the so, suppose this particular frame, it tells that it starts at this particular address. So, with that this displacement will be added and that will come to; so, that that will give me the corresponding address in the physical memory.

So, again; so, this hardware has to be provided, this page table should be there, then this logical address to physical address conversion so, that has to be done. So, how is it done, so that is provided by the memory management unit. So, if we go back now and try to understand, how this paging works. So, page table is kept in memory and the page table has got a page table base register. So, this entry's of this page table is basically, the corresponding frame number. So, page table entry, if a process has got say 4 pages then these page table entries 0, 1, 2, 3.

So, if this first page number 0, if it is loaded at memory frame 10, then page number one may be at 20 page number 2 may be at 22 and page number 3 may be at memory frame 35. So, that means, that if this is the process and it has been divided into four such pages page 0, 1, 2 and 3. Now if you are trying to access a location here, then we know that from the page table that the corresponding memory frame is 10. So it will come to memory frame 10 and this will be accessing somewhere here.

Similarly, if you are trying to access this particular location, then it will be the memory frame 20 and then it will be accessing something here. So, it has to consult this page table the system has to consult this page table to understand what is the corresponding frame number and after getting that frame number, this frame number and the offset so, they have to be combined together to get the corresponding page; corresponding entry in the at physical memory. So, this page table base register so, that is the size of the; that points of the beginning of the page table and this page table length register so, that indicates size of the page table. So, how many pages are there.

So, basically as I like the segmentation policy so, we have seen that the segment able base register value. So, it is kept as part of memory limits information in the PCB similarly, these two information they are to be kept in the P C B for system supporting this pager memory management policy.

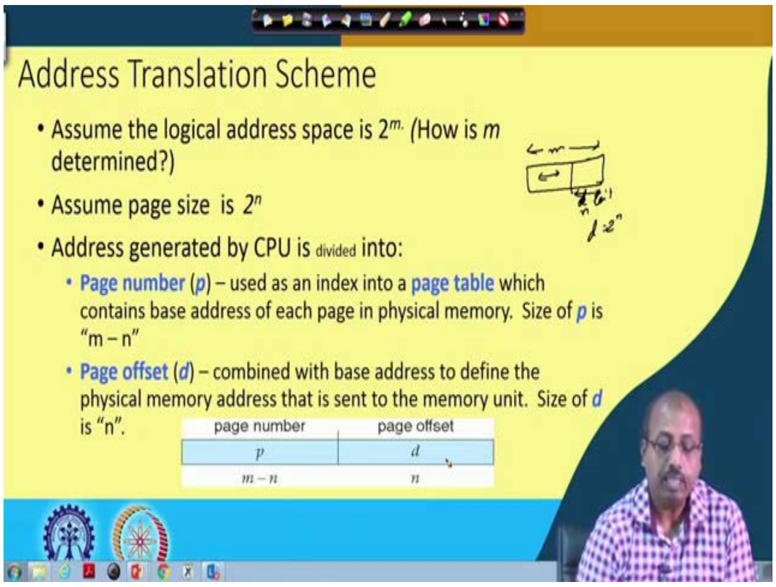
So, this actual page table is in the memory ok. So, this is this is my full memory may be this page table further process is located at this particular region, this particular so, this is the space allocated for page table. So, this address is say 500 in that case this page table base register value; so, this will be equal to 500 and this P T L R so, this particular process has got only four pages. So, P T L R value is equal to 4. So, these two values are actually stored in the PCB of the process in the memory limits portion.

And when this process will make a transition from ready to run at that time this P T B R and P T L R so, these two registers values will be loaded from the P C B and then once they are loaded then the process is ready for execution.

So, it can go into the execution phase. And then whatever logical address is generated by the C P U so, the M M U first consults the P T B R register to locate the page table and after locating the page table from the logical address it takes the page number to consult the page table and come to the corresponding frame number. And after getting the frame number, that frame number and offset that gives the actual memory location.

And this P T L R register value. So, it is use to check, whether the page number that is generated by a process logical address is valid or not. So, if a process tries to access some location which is beyond its address space so, that page number will not match. So, page number will be more than the P T L R value so, as a result it will be going outside the processes address space and then address error trap can be generated. So, this is the policy and we have already discussed that, this paging policy it can still have internal fragmentation, particularly with the last page of the process, but otherwise there is no fragmentation.

(Refer Slide Time: 16:02)



Address Translation Scheme

- Assume the logical address space is 2^m. (How is m determined?)
- Assume page size is 2^n
- Address generated by CPU is divided into:
  - Page number (p) – used as an index into a page table which contains base address of each page in physical memory. Size of p is "m – n"
  - Page offset (d) – combined with base address to define the physical memory address that is sent to the memory unit. Size of d is "n".

page number	page offset
p	d
$m-n$	n

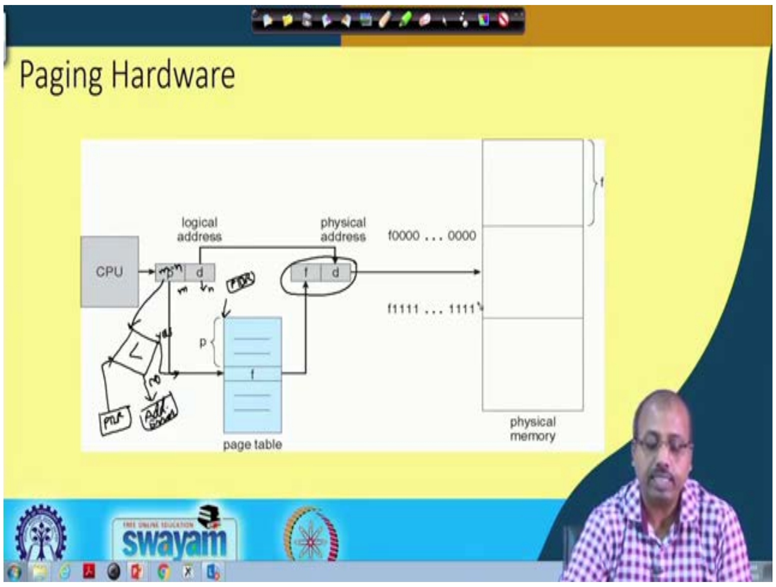
So, this is the address translation scheme. So, assume that the logical address space is two power m, now how this m is determined? So, m is determined by the total number of address lines that the processor has ok. So, if you look into the processor; so, it will tell it has got 64 bit address line or address bus or 32 bit address bus or 16 address bus.

So, that actually determines the value of m and we assume the page size to be a power of 2, 2 power n. So, as I was telling in the previous example, page size is 1000; 4 k so, that is 2 power 12 so, n equal to 12. Now the address; the logical address generated by C P U, it is divided into two parts page number and page offset.

And page offset part is the size of the page and since the page size is 2 power n so, it has to be n bit number. So, this page offset is combined with the base address to define the physical address that is sent to the memory unit. So, size of d is definitely n. And the page number part so, it is the used as an index of the page table. So, basically we have got this address generated by C P U logical address and this logical address is m bit out of that so, this d bits are kept for this offset part and this m minus sorry yes this not d bit so, the n bit. So, a d called to 2 to the power n.

So, this m minus n bit so, that actually gives me the page number part. So, that page number part so, that is used for indexing the page table and come to the corresponding frame number.

(Refer Slide Time: 17:59)



Paging Hardware

So, that is how this translation will be done. So, as I was discussing. So, this whole page logical address that is generated is m bit and out of that. So, d is n bit so, this m minus n p is m minus n bit. So, this m minus n bit p so, that is used to access this page table. So, as I was telling that the page table base register P T B R so, this is actually pointing to this. So, before that you can assume that there is a check here with the P T L R.

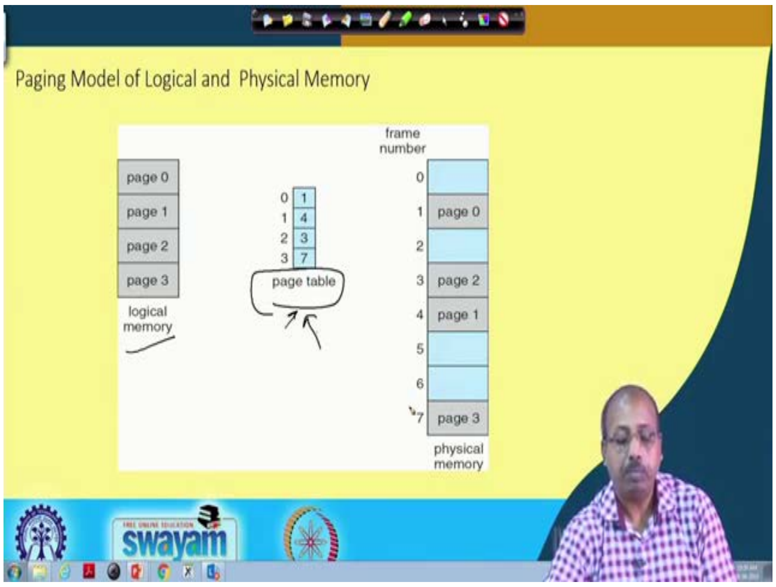
We have got the P T page table length register entry. So, this p that is coming so, this is feed here and if this p is less than the P T L R value, then only this goes this side otherwise so, this is a P T m minus n is less than P T L R. So, if it is yes then it goes like this, if it is no in that case this is at address error; this is an address error.

Now, after this f has been given, after this p has been found to be ok so, this p is used to index into this page table so, that gives us the frame number. And this frame number is used to access the corresponding memory frame and this d value is coming directly. So, that this makes the corresponding physical address space and the physical address and then that physical address goes into the memory.

So, that is how this paging hardware works, but the difficulty that we have is of course, that page table. So, as I said that page table is in memory so, to access one memory locations so, we are actually accessing memory twice. Once for accessing the page table and once for accessing the actual memory location. So, in some sense you can say that the memory access time is doubled ok. Because of the reason that ok, we have to access this page table once and this physical memory next. And memory access time is the largest time in a C P U say in a computer system, because the it has to go outside the C P U. So, this address bus data bus configuration and all those has to be done so, that takes quite some time for the memory to respond. So, that way if you increase 1 memory access time to 2 memory access time for a single memory variable or instruction access.

That is not a very good proposition. So, there are certain techniques by which this can be addressed to this time can be reduce. So, it cannot be eliminated altogether, but this time can be reduced to a significant extent.

(Refer Slide Time: 20:44)



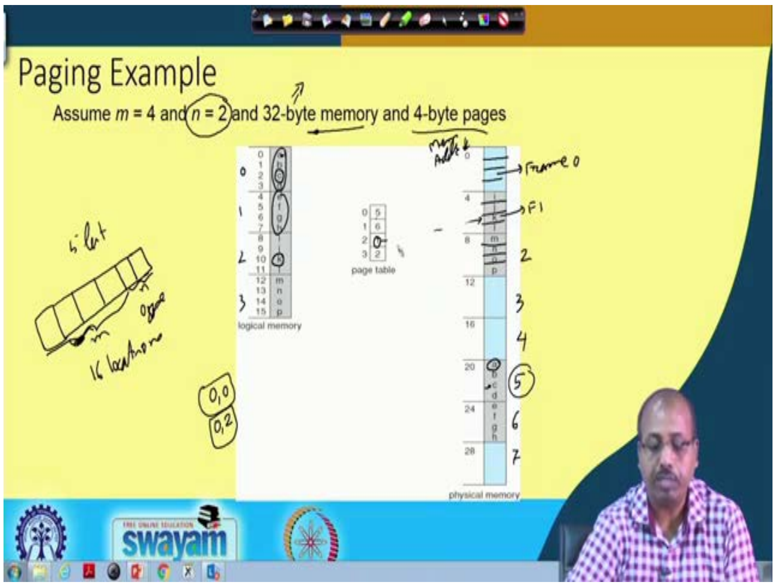
Paging Model of Logical and Physical Memory

Will come to that; so, this paging model of logical and physical memory. So, this logical memory is divided into these pages like this page 0, 1, 2, 3 and then this page table is telling is giving me the corresponding frame numbers for page 0 it is 1 page 1 it is 4 like that. And then they have been so, this where this page table is populated when this program is loaded into the memory ok.

So, they when it is residing in the disk so, it is residing in this format. Now when it is; when the process makes transition from new to ready, the pages are loaded into the memory and if the loader finds that the blocks are entry empty in this fashion and accordingly it has filled up the blocks with that.

So, it will fill up after that; after that the loader will prepare this page table, it will populate the page table with the corresponding frame number entries. And once that is done my program loading is over ok.

(Refer Slide Time: 21:48)



Paging Example

Assume $m = 4$ and $n = 2$ and 32-byte memory and 4-byte pages

So, this is; so, once it is loaded then suppose, we have to have; we have got some m equal to 4 that is there are 16 pages and n equal to 2. So, that is there are the offset is equal to; so, this m and n so, m is the address that the total number of a logical address space and n is the page size.

So, n equal to two so, this each page is of 4 bytes and we have got a 32 byte. So, we have got a 32 byte memory space and we have got 4 byte pages. So, each page is 4 byte so, this is n equal to 2. So, 32 byte means total number of address lines generated by the C P U is 5 bit. So, out of this 5 bit so, first two bits this significant 2 bit so, that constitute the n so, that is the offset part and the remaining 3 bits so, that actually the constitutes the so, this 32 bit so, m equal to 4. So, that is the address space the logical address space generated is yes.

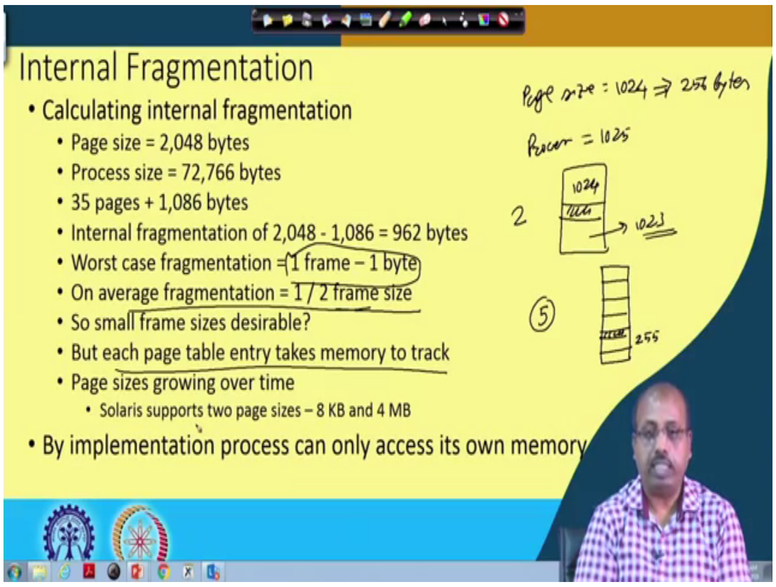
So, that logical address that is of 16 locations and we have got n equal to 2. So, n equal to 2 is that is 2 bits part. So, this is my m part and this is the n part. Now this 0, 1, 2, 3 so, this is the first 4 pages. So, 0, 1, 2, 3, then 4, 5, 6, 7 like that and we have got then the corresponding. So, these are some of the entries in the those pages ok. Now suppose this page number 0 is loaded in the frame number 5. So, page number 0 is page number 5 is not shown here in fact.

So this page number; this page table contains this entries like 5, 6, 1, 2 so, page number 2 will be loaded in memory frame number 5. So, 5 is page number 0 is having a b c d so, that is loaded in 20 actually this should be 20 in that case. So, this is yeah; so, this is frame number 0 yeah this is fine. So, this is frame 0 so, this is frame 1 this is frame 2, 3, 4, 5, 6, 7 so, it is like that. So, these are the memory addresses. So, 0 each page is of 4 byte so, these are the four locations 0, 1, 2, 3, then this is the next four locations then these are the next four locations so, like that.

So, this particular column is the memory address. This particular column is the memory address. So, this 5 is so, page so, this is 5. So, this page number 0 so, if you are trying to access this variable a. So, it says that it is page number 0 and the offset is also 0. So, you see; so, from this it will find out that page number 0 is frame number 5. So, it will come to a frame 5 and then offset 0 so, it will come to variable a. Similarly if it is trying to access the variable c, then it will see that this is page number 0 and the offset is 2. So, it will come to corresponding frame number 5 and offset 2 that is it will come to c.

Similarly, if you find to if it is trying to access say this particular location this k 10 ok; so, memory location logical address 10, then it finds that this is located in this is my page number 0, 1, 2, 3. So, page number 2 is in frame number 1. So, this is frame number 1 so, this is i j k; so, k; so, it will be coming to this particular entry and the offset is 3. So, it will be accordingly accessing so, this particular frame. So, this way this page table can tell us what is the corresponding frame number and from that frame number it can generate the corresponding physical address.

(Refer Slide Time: 26:29)



Internal Fragmentation

- Calculating internal fragmentation
  - Page size = 2,048 bytes
  - Process size = 72,766 bytes
  - 35 pages + 1,086 bytes
  - Internal fragmentation of 2,048 - 1,086 = 962 bytes
  - Worst case fragmentation = 1 frame – 1 byte
  - On average fragmentation = 1 / 2 frame size
  - So small frame sizes desirable?
  - But each page table entry takes memory to track
  - Page sizes growing over time
    - Solaris supports two page sizes – 8 KB and 4 MB
- By implementation process can only access its own memory

Now; so, there can be internal fragmentation for calculating the internal fragmentation, suppose the page size is 2 k; 2048 bytes and the process size is 72766 bytes. So, that way it will have 35 pages, so, if you divide this 72766 by 2048. So, it will have 35 will be the quotient and the remainder will be 1086. So, this 1086 byte is the additional thing that is stored in the last segment. And this; so the for the last page so, this 2048 minus 1086 so, this is the total amount of bytes that are wasted. So, this is 962 bytes. So, that will be wasted.

So, what is the worst case fragmentation? So, worst case fragmentation will come if we have got this thing. So, may be my page size is equal to say 1024 and i have got a process of size say 1025 bytes, then what will happen? So, it will have two pages the first page will have 1024 bytes in it and the next page will have only 1 byte in it so, remaining 1023 bytes will be wasted.

So, this is the worst case situation that can come. So, this 1 frame minus 1 byte so, that is the worst case fragmentation, on an average you can say half of the frame size is the wastage ok. So, that is go because of this internal fragmentation. So, to if you want to reduce this internal fragmentation, and then definitely I would suggest that the frame sizes be small. So, if the frame size is small, then definitely so, half of it; is the average wastage is half of the frame size. So, the frame size is small then the wastage will also be small. So, should we go for small frame size, but if we go for small frame size, then the page table number of entries in the page table so, that will also be large.

So, instead of having say, 1024 bytes page size. So, if I have got page size to be equal to say 256 bytes. Then the wastage will be less because i will have 5 pages in my program now, the first 4 pages will have 256 bytes then the last page will have only 1 byte filled up so, the wastage will be 255 bytes.

So, wastage is reduced from 1023 it has come to 255, but what has happened is that this number of pages for that process has increased. So, previously it was only 2 pages, now it has become 5 pages ok. Now it has become five pages. So, for each page table entry so, you have to remember the corresponding frame number of the memory. So, as a result so, that overhead will also come into picture.

So, each page table entry keeps takes memory to track and page sizes growing over time so that are another option. So, the sometimes we keep the page size to be say 8 kilobyte sometimes to 4 megabyte so, like that. So, as the process size increases may be the page size also increase. So, initially when the process has not consumed lot of stack and dynamic and a heap space. So, maybe we can say we can give it small block size, but as the process is running and it is taking more of space so, the block size may increase. So, naturally the implementation will have its own memory requirement. So, we will continue with this in the next class.