Introduction to Operating Systems
Prof. Chester Rebeiro
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Week – 02
Lecture – 08
Segmentation

Hello. In the previous video we had looked about Memory Management, especially we had focused on a concept known as Virtual Memory. In this video we will look at another important concept for memory management which is known as Segmentation.

(Refer Slide Time: 00:33)

Programs are a collection of logical modules

Logical modules : such as global data,
stack, heap, functions, classes, namespaces, etc.

38

So, now when we look at programs in general they can be split into logical modules. So, logical modules for instance global data, stack, heap, functions, classes, name spaces, and so on.

(Refer Slide Time: 00:47)

Programs are a collection of logical modules

Logical modules : such as global data, stack, heap, functions, classes, namespaces, etc.

Virtual memory does not split programs into logical modules, instead splits programs into fixed size blocks.

38

When we look at virtual memory on the other hand, virtual memory does not split programs into logical modules instead virtual memory actually splits programs into fixed sized blocks. So, while this would work in general it is not a very logical thing to do, for instance we may have a few instructions of a function in one logical block while the rest of the instructions of that function within a totally different logical log.

(Refer Slide Time: 01:17)

Programs are a collection of logical modules

Logical modules : such as global data, stack, heap, functions, classes, namespaces, etc.

Virtual memory does not split programs into logical modules, instead splits programs into fixed size blocks.

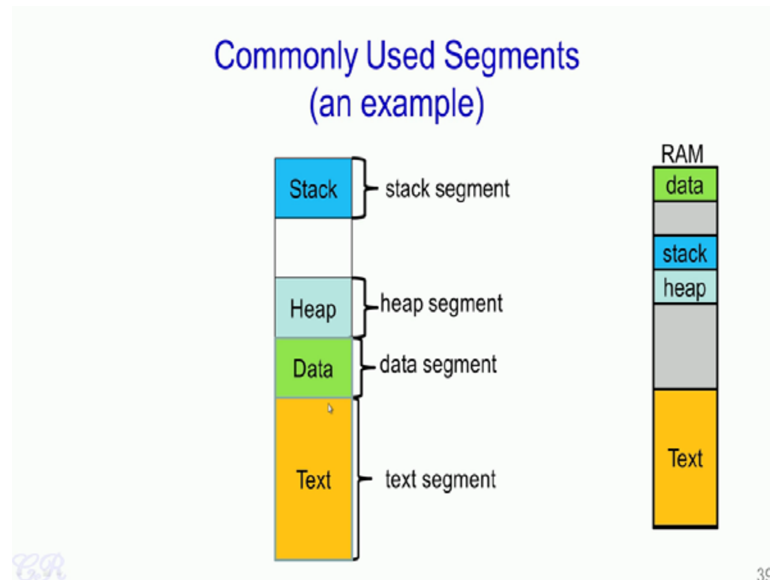
Segmentation can be used to split program into segments that are more logical.

The segment size could range from a few bytes to the maximum size (4GB in 32 bit Intel machines)

38

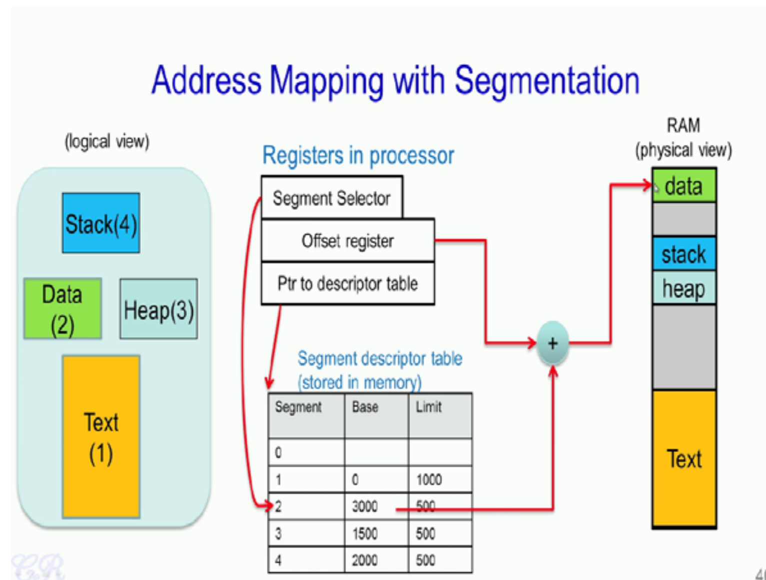
Segmentation on the other hand, achieves a more logical split of the program. So, we could define segments to vary in size from a few bytes to up to 4 Giga Bytes and we could define segments to be in a more logical order. For instance, we could have each function within our program to be in a different segment as shown in this particular slide.

(Refer Slide Time: 01:39)



A very common usage of segmentation is to split the program into various segments such as the text segment data segment heap segment and the stack segment. So, this is known as the logical view of the program.

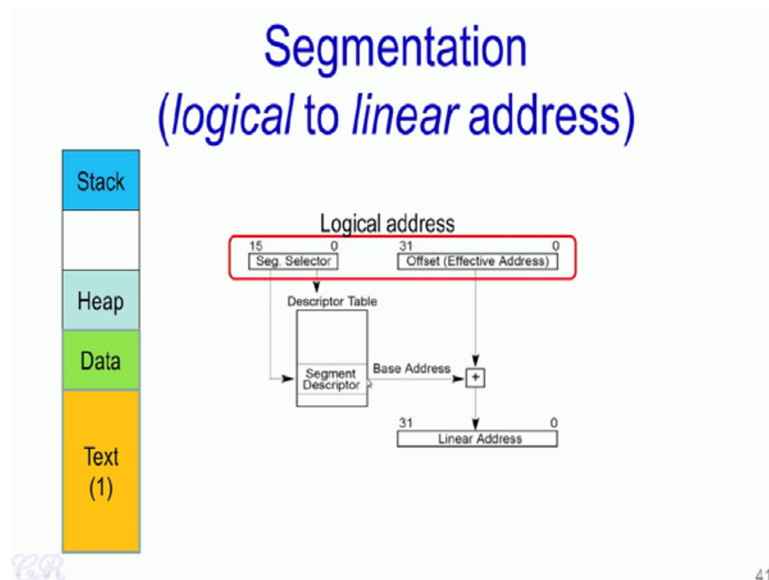
(Refer Slide Time: 01:55)



Now, let us look how the address mapping is done from the logical view to the physical view in segmentation. So, essentially we would have a segment descriptor table which is stored in memory each row in the segment descriptor table pertains to one particular segment. For instance, the data segment 2 is at an offset two in the Segment descriptor table, and the offset would specify the base address in RAM and the Limit of the segment.

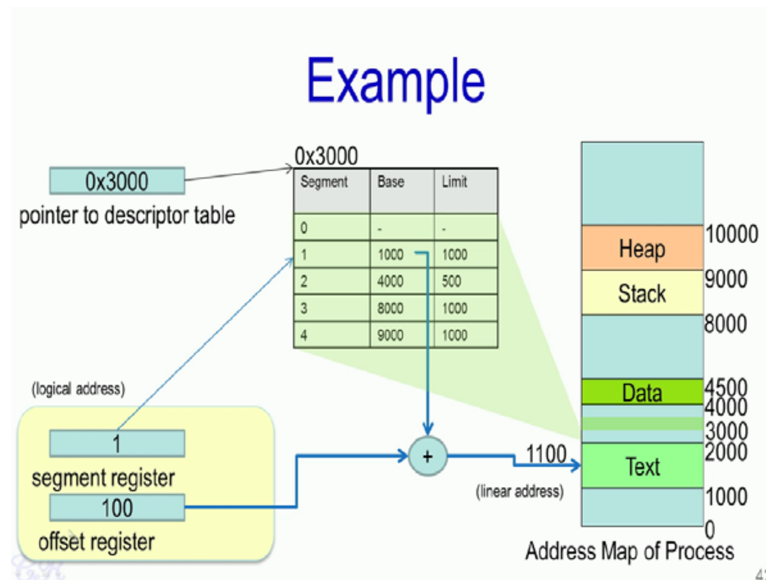
Now, in order to make the mapping the processor would have at least 3 registers that is the segment selector, offset register, and the pointer to the descriptor table. So, as the name suggests the pointer to the descriptor table is a pointer to the memory location which holds the descriptor table. The segment selector on the other hand, is an offset into the descriptor table. The memory management unit in the processor would look up this particular offset and pick the base value 3000. So, this base value is then added with the contents of the offset register to get what is known as the Effective Address. So, this Effective Address will correspond to some address in the RAM.

(Refer Slide Time: 03:16)



Let us look at another view of this mapping scheme. The logical address comprises of two parts, it comprises of a segment selector as well as an offset address also known as the effective address. So, the segment selector in an Intel 32 bit processor is of 16 bits, while the effective address or the offset register is of 32 bits. So, what would happen if the contents of the segment selector are used as an offset into the descriptor table? The memory management unit would then pick up the base address from this particular offset, and add the contents to the effective address to what is known as the linear address.

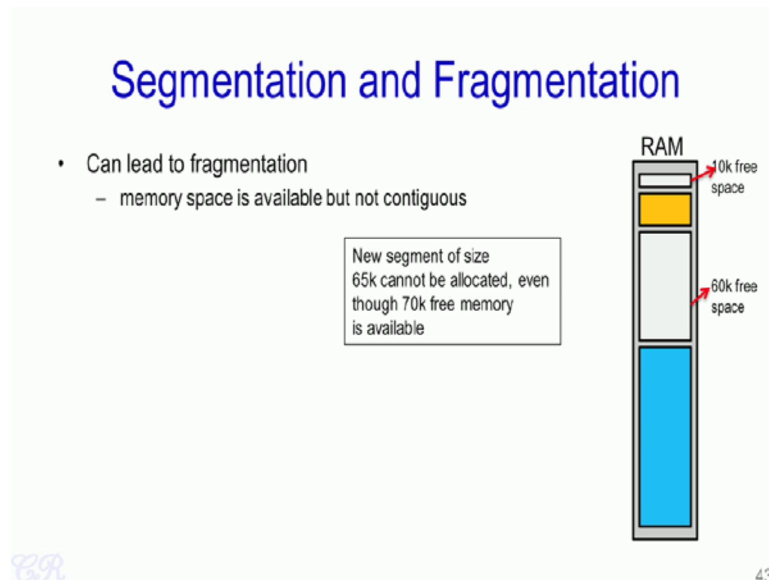
(Refer Slide Time: 03:58)



Let us look at the mapping done in segmentation with an example. Let us say the register containing the pointer to the descriptor table has a value of 3000. So, this means that at an address 3000 in RAM there is the descriptor table which contains the mapping for the various segments. Now let us say that the segment register has a value 1, so this means we are trying to use the segment at offset 1 in the descriptor table.

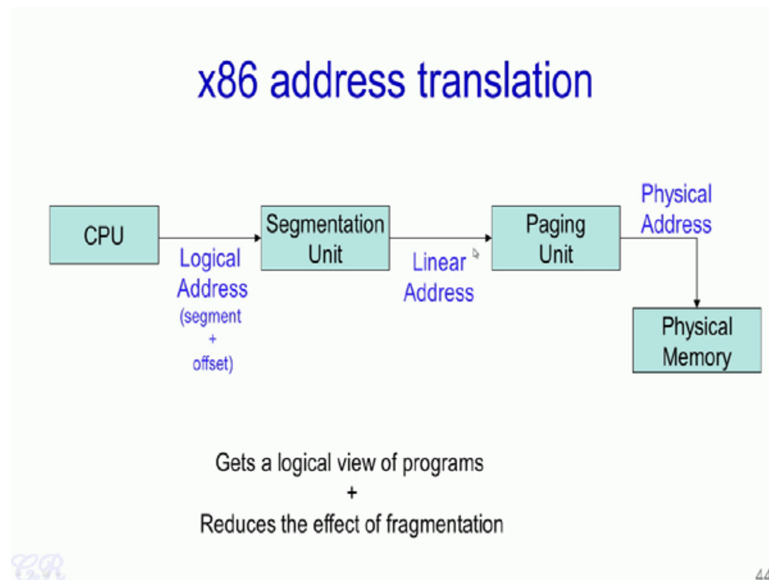
The memory management unit would then take the base address corresponding to this offset, which is 1000 in this case and use the offset register which has a value of 100 to get what is known as the linear address that is 1100. Now, the segment registers along with the offset register form what is known as Logical Address.

(Refer Slide Time: 04:51)



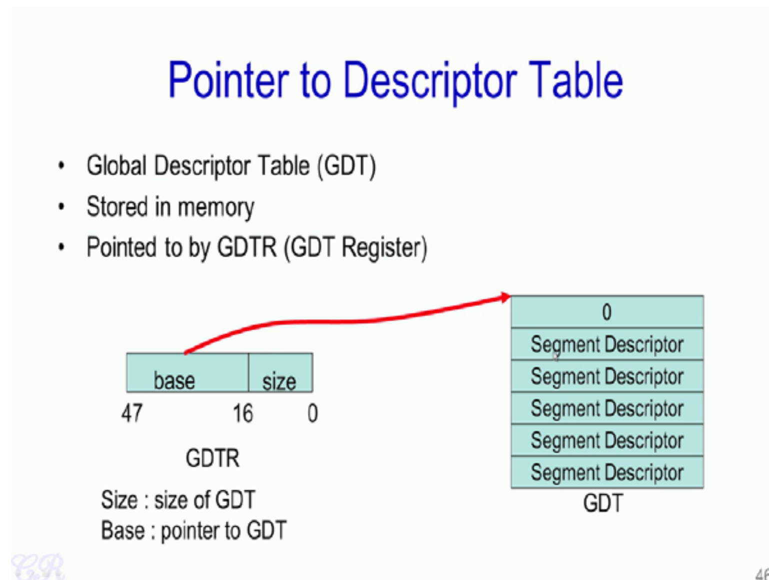
So, one of the biggest problems with segmentation is Fragmentation. Let us look at this particular example; we have 70 kilobytes of space which is free in the ram. However, this free memory is not in contiguous locations we have 60 kilobytes of free space in one chunk, another 10 kilobytes of free space in another chunk. So, this cannot be used to allocate a new segment which is of 65 kilobytes. So, even though there is 70 kilobytes of free memory available, the memory is not in contiguous locations and therefore, cannot be used. Fragmentation is one of the biggest limitations of segmentation; however, fragmentation is much less an issue with virtual memory.

(Refer Slide Time: 05:41)



We will now look at how Intel x86 systems makes use of both segmentation as well as paging, so that advantages of both are obtained. So, in an x86 system, the CPU generates a logical address comprising of a segment plus an offset. This is sent to a segmentation unit which then generates a linear address. The paging unit is essentially the virtual memory management which would take the linear address and generate the physical address. So, let us see how the segmentation unit is designed in x86 systems.

(Refer Slide Time: 06:16)



X86 systems have 2 types of descriptor tables. So, one is known as the Local Descriptor Table, while the other is known as the Global Descriptor Table which we will be presenting over here.

The global descriptor table is stored in memory and has a format as shown over here. So, essentially it has the first field which is 0, followed by Segment Descriptors. This particular global descriptor table is pointed to by a register known as the Global Descriptor Table Register or GDTR. The GDTR is a 48 bit register, having the following format. The least significant 16 bits contains the size of the GDT, while the upper bits contain the base address that is the pointer to the GDT; that is this pointer. So, let us look at what the content of the segment descriptor is.

(Refer Slide Time: 07:06)

Segment Descriptor

- Base Address
  - 0 to 4GB
- Limit
  - 0 to 4GB
- Access Rights
  - Execute, Read, Write
  - Privilege Level (0-3)

Access	Limit
Base Address	

47

So, the segment descriptor contains 3 parts it has a Base Address, it has a Limit and it has an Access Rights. The Base Address and the Limit can take values from 0 to 4 Giga Bytes, while the access rights are bits which specify various access policies such as Execute, Read, Write or the Privilege Level, for that particular segment.

(Refer Slide Time: 07:30)

Segment and Offset Registers

- Holds 16 bit segment selectors
  - Points to offsets in GDT
- Offset registers are 32 bit registers
- Segments associated with one of three types of storage
  - Code
    - %CS register holds segment selector
    - %EIP register holds offset
  - Data
    - %DS, %ES, %FS, %GS registers hold segment selector
  - Stack
    - %SS register holds segment selector
    - %SP register holds stack pointer

48

Next, let us look at the segment and offset registers in Intel 32 bit machines. So, the segment selector registers are 16 bit segment selectors which points to offsets in the GDT. The offset registers are 32 bit registers. So, quite often the segment selectors, a couple bit corresponding offset registers. For instance, in order to access the code segment we use the CS register which is the segment selector for the code segment, and the corresponding EIP register which is the offset register known as the Instruction Pointer.

In order to access the Data segment we have several segment registers such as the DS, ES, FS and GS. In order to access the Stack segment we have the SS register which holds the segment selector, and the SP register which holds the stack pointer. All these segment selector registers and offset registers along with the GDTR and the GDT table present in memory are used to convert the logical address to a corresponding linear address.

Next we will look at paging unit which essentially manages the virtual memory mapping in the x86 system. So, the paging unit takes a linear address and converts that to an equivalent physical address which is then used to address the physical memory or the RAM.

(Refer Slide Time: 08:58)

Linear to Physical Address

- 2 level page translation

Linear Address
Dir | Table | Offset
Physical Address
Offset
Flags
Page Table
Flags
Page Directory
CR3

- How many page tables are present?
- What is the maximum size of the process' address space?

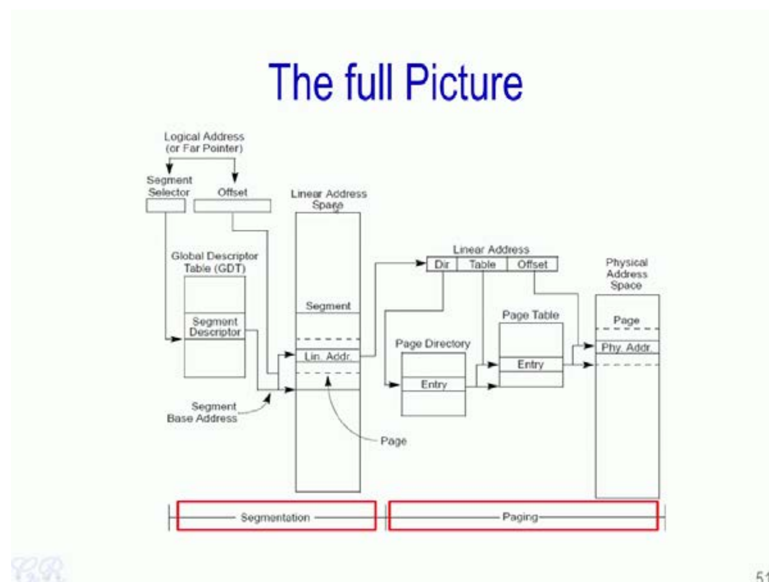
ref : mmu.h (PGADDR, NPDENTRIES, NPTENTRIES, PGSIZE)

50

The paging unit in x86 system comprises of two level page translation. It takes a 32 bit linear address which is split into 3 parts, the most significant 10 bits is known as the directory entry, followed by which there is 10 bits for the table index and finally, the least significant 12 bits are the offset. The directory entry points to a particular offset in the page directory. The page directory is a special table which is present in the RAM and it is pointed to by the CR3 register. The contents of the page directory point to a particular page table and an offset within that page table is taken from the table index. the contents of this particular page table along with the offset are then used to form the physical address.

I have two questions for you. One is how many page tables are present? How many of such page tables are present in a 32 bit Intel system? While the second question is what is the maximum size of the processes address space? So, given that each process has such a linear address to physical address mapping. So, I want you to actually find out what would be the maximum size of a processes address space.

(Refer Slide Time: 10:18)



This particular slide shows the full address translation in an x86 system. The CPU puts out a logical address comprising of a segment selector and an offset. The segment selector is an index into the global descriptor table into something known as the segment

descriptor. The segment descriptor along with the offset then creates what is known as the linear address, and this entire space is known as the Linear Address Map for the process. The linear address comprises of 3 components that is the Directory entry, the Table index and the Offset. So, the directory entry indexes into the page directory and this content is then you used to select a page table. The table index is used to get an offset within that page table and the contents of this, along with or the final 12 bits in the linear address is then used to obtain the final physical address which is used to read or write data to the RAM.

With these set of videos we had looked at memory management schemes such as virtual memory and segmentation and we have seen how Intel manages address translation in 32 bit systems.

Thank you.