

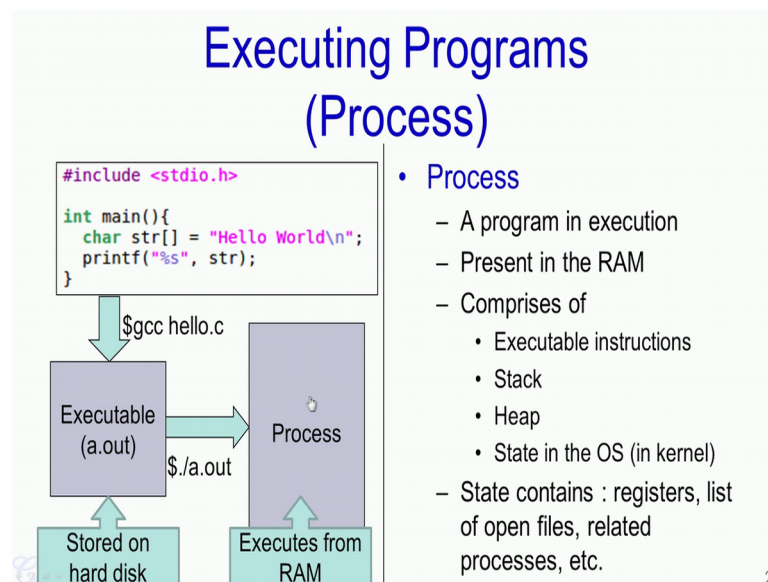
**Introduction to Operating Systems**  
**Prof. Chester Rebeiro**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Week - 02**  
**Lecture - 05**  
**Memory Management**

Hello. In a previous video (Lecture 4 video) we have seen, how the operating system has to manage the CPU because it is probably the most important resource in the system. Arguably, the second most important resource, rather the second most important hardware resource in the system is the Memory.

In this video, we will look at how the operating system manages the memory? Essentially, we will see how the operating system manages the RAM present in the system.

(Refer Slide Time: 00:56)

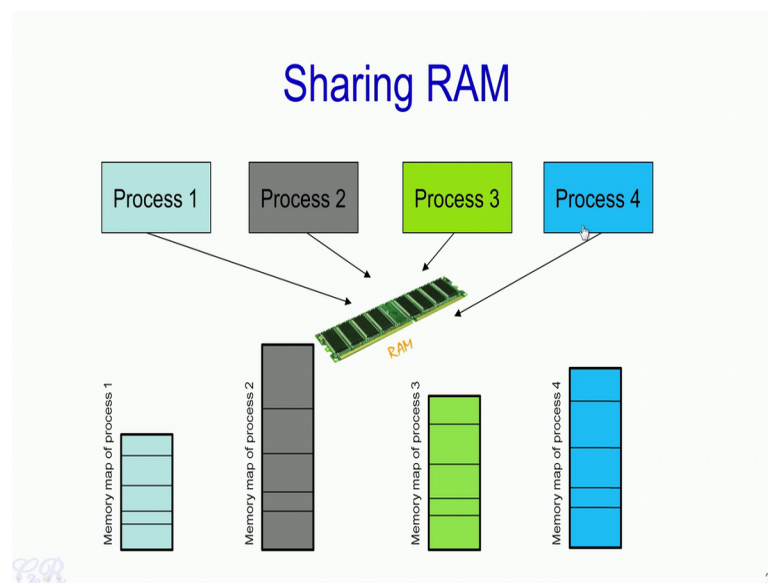


So let us review this particular slide (mentioned above) again. We have seen that when we take up a particular program and this could be any program written in any language and when we compile it, we will get what is known as an Executable.

Now, the executable will be stored on the hard disk and whenever a user runs this program or executes this program (mentioned in above slide), it creates what is known as a Process. So, this process is created by the operating system and it executes in RAM. So, essentially what the operating system would do is that the executable stored in the hard disk, would be loaded into the RAM and then on execution is passed to this particular program and this program will then execute in the CPU.

So, as we have seen this particular process (mentioned in above slide) which is present in memory comprises of several segments, such as the text segment which contains the Executable instructions, the Stack, Heap and also as we have seen some hidden metadata in the operating system such as registers, list of open files and related processes. So, all these actually constitute the process. Now, what is important for us is this process (mentioned in above slide) and it is presents in the RAM.

(Refer Slide Time: 02:45)



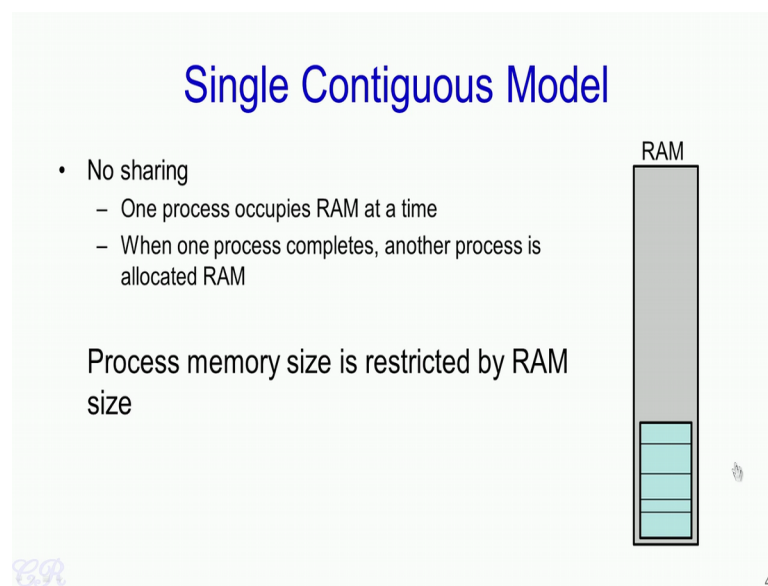
So as we know the RAM or Random Access Memory also called as the Main Memory is a limited resource in the system. So, each system would have something like 4, 8, 16 or 32 GB of RAM. At the same time we may have multiple processes executing almost simultaneously. So, like we have seen in the previous video (Lecture 4 video), these

processes may execute in a time sliced manner and if there are multiple processors present in the system, then these processors could also execute in parallel.

However, in all these cases, these processes and their corresponding memory map should be present in the RAM. Essentially the memory map corresponding to process 1, the memory map corresponding to process 2, memory map of process 3 and the memory map of process 4 should be present in the RAM, in order that these 4 processes execute in parallel or in a multi tasked environment.

Now, what we are going to see is how the operating system manages this resource (mentioned in above slide) or in other words the RAM such that it would allow multiple processes to execute almost simultaneously in the system.

(Refer Slide Time: 04:25)



So, one of the most primitive ways of managing memory especially done for the older operating systems is what is known as the Single Contiguous Model. So, essentially in this we have a RAM over here (mentioned in above slide) which is the RAM of the system and what is ensured or what is done by the operating system is that this RAM is occupied by one process at a time. Essentially at any particular instant there is only one


process and it is memory map that is present in the RAM. So, after this process completes executing will the next process will be loaded into the RAM.

(Refer Slide Time: 05:21)

### Single Contiguous Model

- No sharing
  - One process occupies RAM at a time
  - When one process completes, another process is allocated RAM

Process memory size is restricted by RAM size



RAM

4

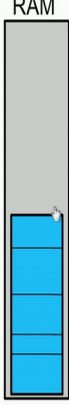
So, the drawbacks are quite obvious.

(Refer Slide Time: 05:25)

### Single Contiguous Model

- No sharing
  - One process occupies RAM at a time
  - When one process completes, another process is allocated RAM

Process memory size is restricted by RAM size

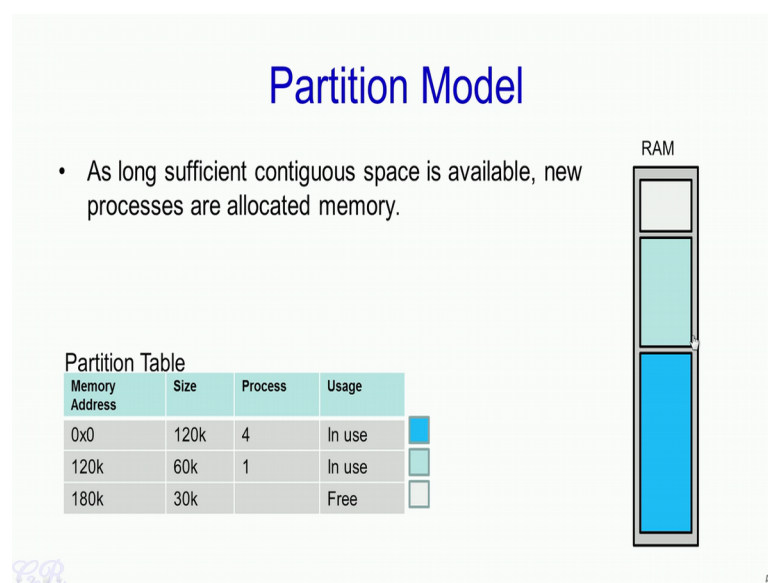


RAM

4

The first is that we are forced to have a very sequential execution. When one process completes, only then the second process could occupy the RAM and so on. Another limitation of this particular model that is a single contiguous model is that the size of the process is limited by the RAM size. For instance, let us say we have a RAM which is of say 12 KB while this seems to be a very small amount of RAM, this size of RAM is quite common in embedded system. So, given this RAM of say 12 KB and let us say our process size is of 100 KB then it is quite obvious that this process cannot execute using this RAM. Essentially the RAM is not sufficient to hold the entire process.

(Refer Slide Time: 06:35)



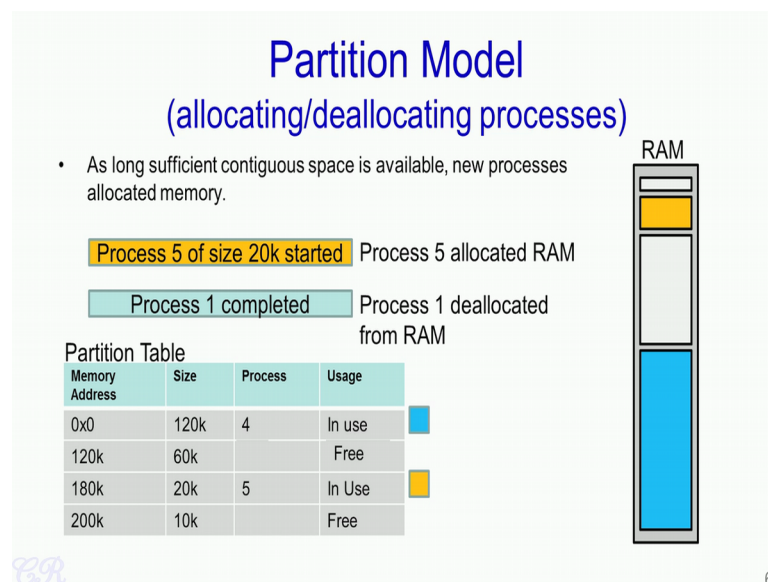
The next model that we will see, which is a slight improvement over the single contiguous model is what is known as a Partition Model. Essentially in this model at any instant of time, we could have multiple processes that occupy the RAM simultaneously. For instance, in this particular case we have two processes. This blue process and the green process (mentioned in above slide) that occupy the RAM and therefore, the processor could then execute this process as well as this process either in parallel or in a time sliced manner.

Now, in order to manage such a partitions, the operating system would require something known as a Partition Table. So, typically this partition table (mentioned in slide above)

would also be present in the RAM. It will be present in an area which is not shown over here. So, the partition table would have the base address of a process, the size of the process and a process identifier. For instance, the blue process has a memory or a base address of 0x0 indicating that is starting from 0<sup>th</sup> location in the RAM and this process has a size of 120 KB. So, 120 KB means up to this point.

There is also a flag known as the usage flag which mentions whether this particular area in RAM is in use or it is free. For instance, let us take process 1 that is this one, the green one over here shown over here (mentioned in above slide). So, this process 1 starts at the memory location 120K that is this point and has a size of 60K. So, it extends up to 180K and this area is also in use. Now, there is another entry in the partition table which is specified as free. So, this starts at 180K and extends for a size of 30K. So, this corresponds to this white area over here (mentioned in above slide). So, the operating system could possibly use this free memory to run perhaps the third process and therefore, would be able to have three processes present in the RAM at the same time.

(Refer Slide Time: 09:34)

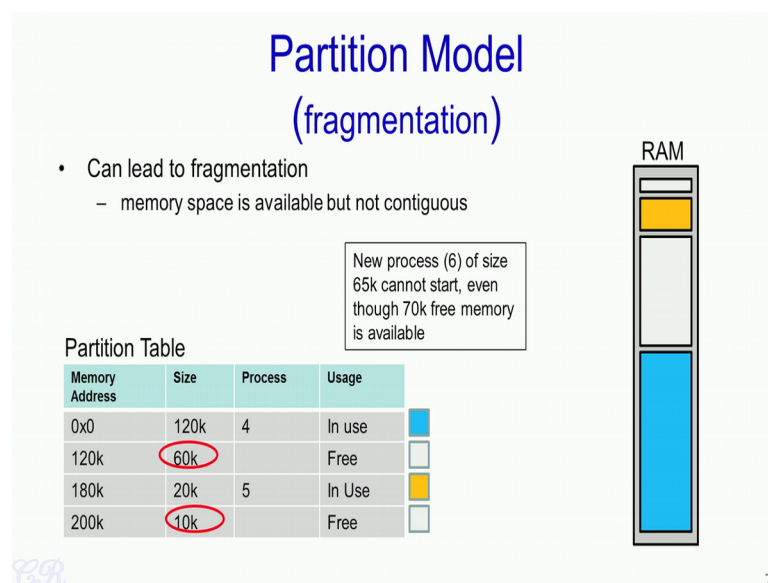


So, let us say a new process with the identifier 5 has just being created. So, the operating system would look into the partition table and create a particular entry for this new process. Now, since this process requires a size of 20 KB of RAM therefore, the

operating system will allocate 20 KB of RAM for that process. So, this allocation is done from the free space and as we see over here (mentioned in above slide) the 3<sup>rd</sup> process now gets end present in the RAM while the free space reduces in size. So, we now have the 10 KB of free space.

Similarly, when a process completes execution, for instance when process one completes its execution, the area in RAM that it holds will be de-allocated. Consequently the entry corresponding to the partition table will be free. So, this would lead to a free memory of 60K in the RAM. So, this corresponds to this particular area in the RAM (mentioned in above slide) which was used by process 1. While this technique of partitioning the RAM and therefore, allowing multiple processes to be allocated in the RAM is quiet easy to do. It could lead what to something known as Fragmentation.

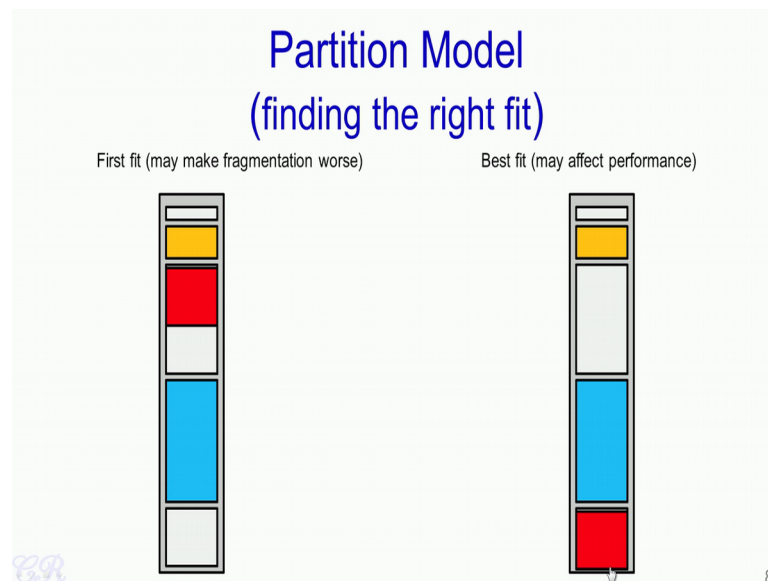
(Refer Slide Time: 11:18)



So, let us say this is how our RAM looks and as we seen we have a total of 70 KB of RAM which is free i.e we have 60 KB + another 10. So, that is 70 KB of free RAM memory. However, in spite of having 70 KB of RAM, a new process such as a new process with Id 6 which has a size 65 k cannot start even though 70 KB of free RAM is available and the reason for this is that the 70 KB of free memory is not in contiguous locations.

For instance, we have 60 KB of free memory present here and 10 KB of free memory present here (mentioned in above slide). So, individually each of these blocks of free memory is not sufficient to start a new process of 65 KB. So, this is what is termed as fragmentation and could result in what is known as the under utilization of the RAM.

(Refer Slide Time: 12:37)



Now, let us assume that when a new process arrives, there is sufficient amount of free spaces that are present in the RAM memory. So, next the operating system has to decide that which among this free memory should the new process be loaded into. For instance over here, there is a new process which has just arrived and we have 3 blocks of free memory, this one, this one and this one (mentioned in above slide). So, which of these three blocks of free memory in the RAM should the new process be loaded into?

There could be several algorithms to achieve this. So, one of the most simplest algorithm is something known as the First Fit. So, with this First fit algorithm what the operating system would do is scan the free blocks in RAM starting from the top and use the first available free block which is at least as large as the processes requirements. So for instance over here, this RAM is scan from the top and it was see that the first free block is too small for the process to fit in. Therefore, it is not allocated here.

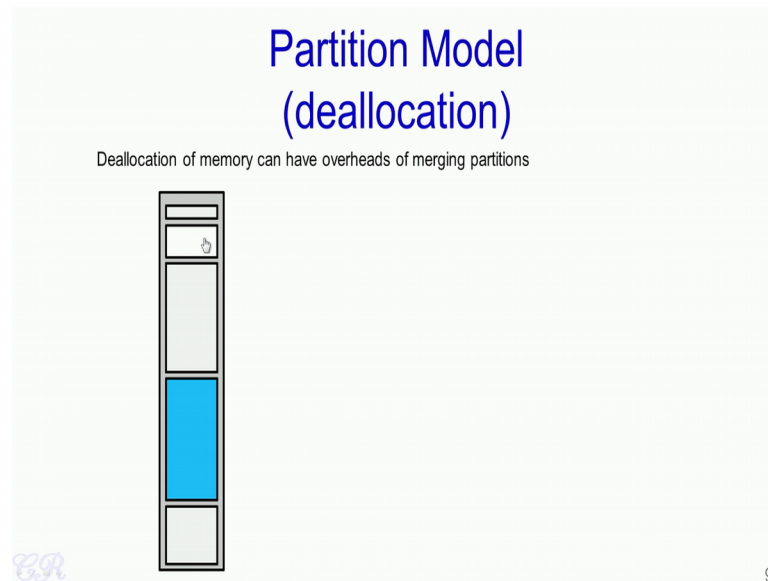


The next free block is large enough to allocate this process. Therefore, the process is allocated here (mentioned in above slide). While this First fit allocation algorithm is extremely easy to perform from an OS perspective, it could make fragmentation worse. Essentially with the First fit, what we are doing is that we are breaking the amount of free blocks into smaller and smaller chunks. Thus individually each chunk will not be able to get a single process, thus making the fragmentation worse.

The other algorithm that we will see is known as the Best fit algorithm and this performs considerably well with respect to the fragmentation. Essentially it will not fragment the memory as much as the first fit algorithm. So, what happens here is that when a new process enters or is created, the operating system will scan through all free blocks that are available and choose the block which is the best fit for that process. So, in this particular case as we have seen earlier as well, this particular free block is too small (mentioned in above slide) to cater to this new process. This free block is too large while the 3<sup>rd</sup> free block is just correct.

So, the operating system would allocate this process into this free block as shown over here (mentioned in above slide). While the best fit algorithm efficiently utilizes the available RAM reducing the fragmentation issue, there may be a performance hit. Essentially, it may result in a deterioration of performance. The reason being is that now the operating system has to scan through every free block that is available and needs to make a decision about which free block is best for the new process and this would take some time.

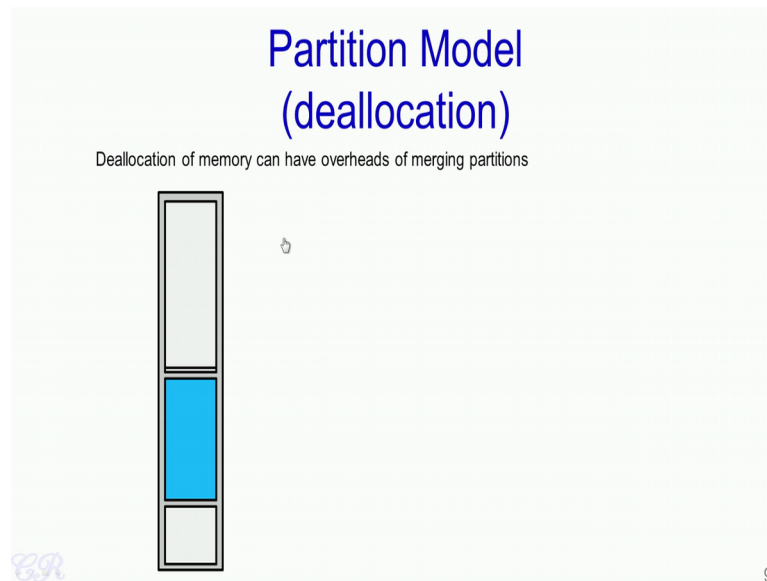
(Refer Slide Time: 16:28)



Another issue with the partitioning model is the case of deallocation. So, essentially deallocation would occur when a process completes its execution or is terminated and has to be removed from the RAM in order to allow a new process or another process to execute from the RAM.

So, deallocation would require that a new free block to be created and it would result in the free flag set corresponding to this block in the partition table. So, what the operating system now has to do is that it should detect that there are indeed three contiguous free blocks that are present in the RAM and as a result of this, it should detect these three contiguous free blocks (mentioned in above slide) and be able to merge these three blocks in to one single block.

(Refer Slide Time: 17:34)



The advantage of merging it into one thick free block is that now this larger free block could cater to a larger process and thereby reducing the effect of fragmentation.

(Refer Slide Time: 17:51)

### Limitations

- Entire process needs to be in RAM
- Allocation needed to be in contiguous memory
- These led to
  - Fragmentation
  - Limit the size of process by RAM size
  - Performance degradation
    - Due to book keeping
    - Managing partitions

Luckily for us, modern day operating systems do much better in managing memory using two concepts  
**Virtual Memory and Segmentation**

10

So, thus we have seen that the major limitation of the single contiguous model as well as the partition model is that the entire memory map of the process needed to be present in

RAM during its entire execution. So, all allocation for the entire process needed to be in contiguous memory and because of these issues, it had led to fragmentation, limit on the size of the process due to base on depending on the RAM size and also performance degradation due to book keeping and also the management of partitions. So, luckily for us modern day operating systems do much better in managing memory. So, most modern day operating systems use two concepts that is virtual memory and segmentation.

So in the next few videos, we will look into these memory concepts rather these memory management concepts and we will also see how the Intel x86 processors manage memory.

Thank you.