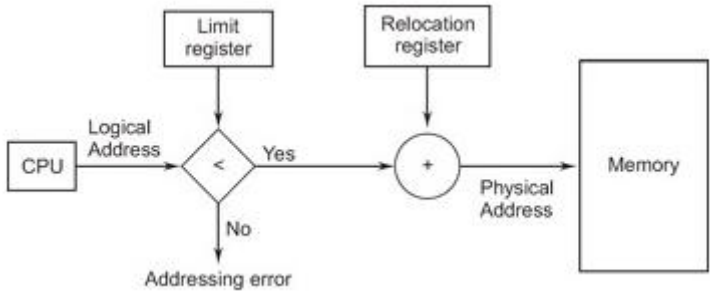
Memory Allocation Schemes

Before discussing memory allocation further, we must discuss the issue of memory mapping and protection. When the CPU scheduler selects a process for execution, the dispatcher loads the relocation and limit registers with the correct values as part of the context switch. Because every address generated by the CPU is checked against these registers, we can protect both the operating system and the other users' programs and data from being modified by this running process.

The relocation-register scheme provides an effective way to allow the operating-system size to change dynamically. This flexibility is desirable in many situations.For example, the operating system contains code and buffer space for device drivers.If a device driver[or other operating-system service] is not commonly used,we do not want to keep the code and data in memory,as we might be able to use that space for other purposes.Such code is sometimes called transient operating-system code;it comes and goes as needed.Thus,using this code changes the size of the operating system during program execution.

To protect the operating system code and data by the user processes as well as protect user processes from one another using relocation register and limit register.

This is depicted in the figure below:

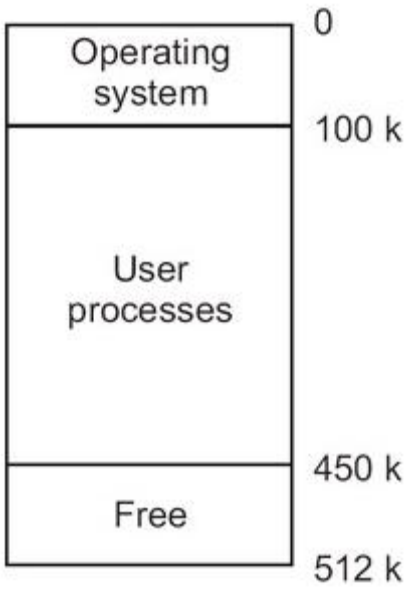


Memory Allocation

Now we will discuss about the various memory allocation schemes.

Single Partition Allocation

In this scheme Operating system is residing in low memory and user processes are executing in higher memory.



Advantages

- It is simple.
- It is easy to understand and use.

Disadvantages

- It leads to poor utilization of processor and memory.
- Users job is limited to the size of available memory.

Multiple-partition Allocation

One of the simplest methods for allocating memory is to divide memory into several fixed sized partitions. There are two variations of this.

- **Fixed Equal-size Partitions**
  It divides the main memory into equal number of fixed sized partitions, operating system occupies some fixed portion and remaining portion of main memory is available for user processes.

Advantages

- Any process whose size is less than or equal to the partition size can be loaded into any available partition.
- It supports multiprogramming.

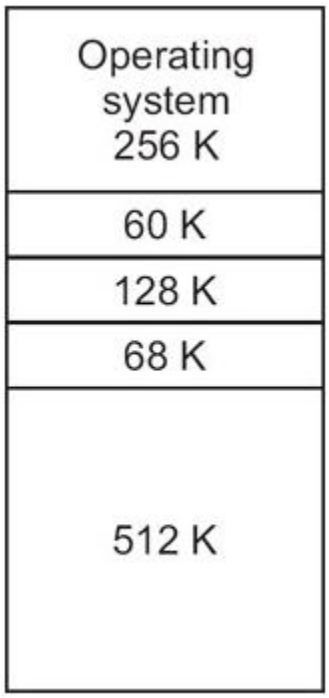
Disadvantages

- If a program is too big to fit into a partition use overlay technique.
- Memory use is inefficient, i.e., block of data loaded into memory may be smaller than the partition. It is known as internal fragmentation.

- **Fixed Variable Size Partitions**

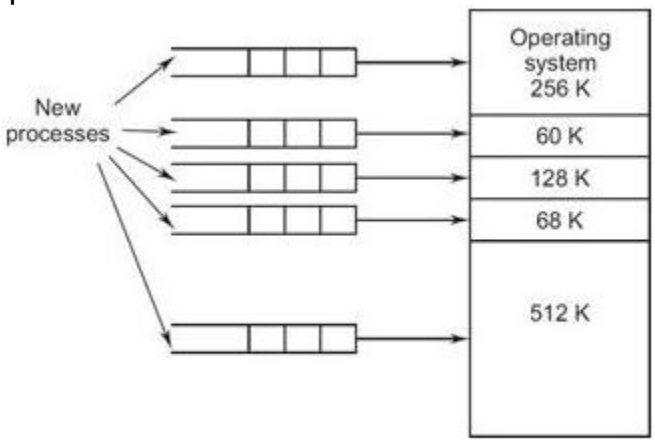
  By using fixed variable size partitions we can overcome the disadvantages present in fixed equal size partitioning. This is

  shown in the figure below:



  With unequal-size partitions, there are two ways to assign processes to partitions.

- Use multiple queues:- For each and every process one queue is present, as shown in the figure below. Assign each process to the smallest partition within which it will fit, by using the scheduling queues, i.e., when a new process is to arrive it will put in the queue it is able to fit without wasting the memory space, irrespective of other blocks queues.



Advantages

- Minimize wastage of memory.

Disadvantages

- This scheme is optimum from the system point of view. Because larger partitions remains unused.
- Use single queue:- In this method only one ready queue is present for scheduling the jobs for all the blocks irrespective of size. If any block is free even though it is larger than the process, it will simply join instead of waiting for the suitable block size.It is depicted in the figure below:

Advantages

- It is simple and minimum processing overhead.

Disadvantages

- The number of partitions specified at the time of system generation limits the number of active processes.

- Small jobs do not use partition space efficiently.

Dynamic Partitioning

Even though when we overcome some of the difficulties in variable sized fixed partitioning, dynamic partitioning require more sophisticated memory management techniques. The partitions used are of variable length. That is when a process is brought into main memory, it allocates exactly as much memory as it requires. Each partition may contain exactly one process. Thus the degree of multiprogramming is bound by the number of partitions. In this method when a partition is free a process is selected from the input queue and is loaded into the free partition. When the process terminates the partition becomes available for another process. This method was used by IBM's mainframe operating system, OS/MVT (Multiprogramming with variable number of tasks) and it is no longer in use now.

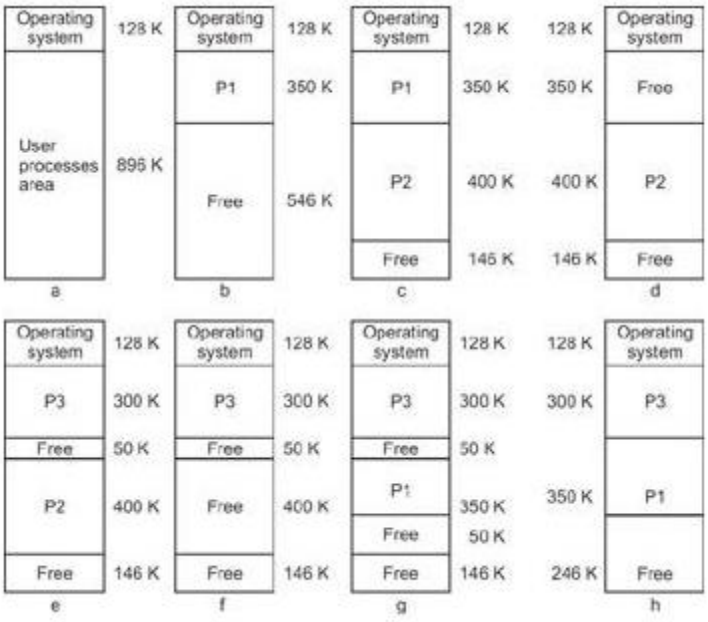
Let us consider the following scenario:

Code:

```
Process|Size (in kB)|Arrival time (in milli sec)|Service time (in milli sec)
P1     |350         |0                          |40
P2     |400         |10                         |45
P3     |300         |30                         |35
P4     |200         |35                         |25
```

Figure below is showing the allocation of blocks in different stages by using dynamic partitioning method. That is the available main memory size is 1 MB. Initially the main memory is empty except the operating system shown in Figure a. Then process 1 is loaded as shown in Figure b, then process 2 is loaded as shown in Figure c without the wastage of space and the remaining space in main memory is 146K it is free. Process 1 is swaps out shown in Figure d for allocating the other higher priority process. After allocating process 3, 50K whole is created it is called internal fragmentation, shown in Figure e. Now process 2 swaps out shown in Figure f. Process 1 swaps in, into this block. But process 1 size is only 350K, this leads to create a whole of 50K shown in Figure g.

Like this, it creates a lot of small holes in memory. Ultimately memory becomes more and more fragmented and it leads to decline memory usage. This is called ‘external fragmentation’. To overcome external fragmentation by using a technique called "compaction". As the part of the compaction process, from time to time, operating system shifts the processes so that they are contiguous and this free memory is together creates a block. In Figure h compaction results in a block of free memory of length 246K.



Advantages

- Partitions are changed dynamically.
- It does not suffer from internal fragmentation.

Disadvantages

- It is a time consuming process (i.e., compaction).
- Wasteful of processor time, because from time to time to move a program from one region to another in main memory without invalidating the memory references.

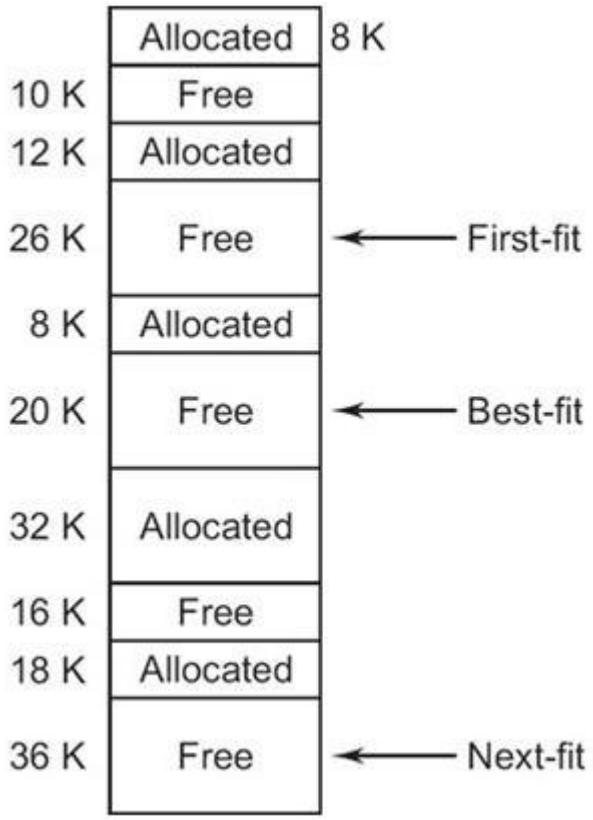
Placement Algorithm

If the free memory is present within a partition then it is called "internal fragmentation". Similarly if the free blocks are present outside the partition, then it is called "external fragmentation". Solution to the "external fragmentation" is compaction.

Solution to the "internal fragmentation" is the "placement" algorithm only.

Because memory compaction is time-consuming, when it is time to load or swap a process into main memory and if there is more than one free block of memory of sufficient size, then the operating system must decide which free block to allocate by using three different placement algorithms.

This is Shown in the figure below:



- Best-fit:- It chooses the block, that is closest in size to the given request from the beginning to the ending free blocks.We must search the entire list, unless it is ordered by size. This strategy produces the smallest leftover hole.
- First-fit:- It begins to scan memory from the beginning and chooses the first available block which is large enough.Searching can start either at the beginning of the set of blocks or where the previous first-fit search ended. We can stop searching as soon as we find a free block that is large enough.
- Worst-fit:- It allocates the largest block. We must search the entire the entire list,unless it is sorted by size. This strategy produces the largest leftover hole,which may be more useful than the smaller leftover hole from a best-fit approach.
- last-fit:- It begins to scan memory from the location of the last placement and chooses the next available block. In the figure below the last allocated block is 18k, thus it starts from this position and the next block itself can accommodate this 20K block in place of 36K free block. It leads to the wastage of 16KB space.

First-fit algorithm is the simplest, best and fastest algorithm. Next-fit produce slightly worse results than the first-fit and compaction may be required more frequently with next-fit algorithm. Best-fit is the worst performer, even though it is to minimize the wastage space. Because it consumes the lot of processor time for searching the block which is close to its size.

Fragmentation

Both the first-fit and best-fit strategies for memory allocation suffer from **external fragmentation**. As the processes are loaded and removed from memory, the free memory space is broken into little pieces. External fragmentation exists when there is enough total memory space to satisfy a request, but the available spaces are not contiguous. Storage is fragmented into a large number of small holes. This fragmentation problem can be severe. In the worst case we could have a block of free or wasted memory between every two processes. If all these small pieces of memory were in one big free block instead, we might be able to run several more processes.

Whether we are using the first-fit or best-fit strategy can affect the amount of fragmentation. First-fit is better for some systems and best-fit is better for others. Another factor is which end of a free block is allocated. No matter which algorithm is used external fragmentation will be a problem.

Depending on the total amount of memory storage and the average process size, external fragmentation may be a minor or a major problem. Statistical analysis of first fit for instance reveals that even with some optimization given N allocated blocks, another 0.5N blocks will be lost to fragmentation. that is one-third of memory may be unusable. This property is known as the **50-percent rule**.

Memory fragmentation can be internal as well as external. Consider a multiple-partition allo9cation scheme with a hole of 18,464 bytes. If we allocate exactly the requested block we are left with a hole of 2 bytes. The overhead to keep track of this hole will be substantially larger than the hole itself. The general approach to avoid this problem is to break the physical memory into fixed-sized blocks and allocate memory in units based on block size. With this approach , the memory allocated to a process may be slightly larger than the requested memory. The difference between these two numbers is**internal fragmentation**- memory that is internal to a partition but is not being used.

One solution to the problem of external fragmentation is **compaction**. The goal is to shuffle the contents so as to place all free memory together in one large block. Compaction is not always possible, however if relocation is static and is done at assembly or load time, compaction cannot be done. Compaction is possible only if relocation is dynamic and is done at execution time. If addresses are relocated dynamically, relocation requires only moving the program and data and then changing the base register to reflect the new base address. When compaction is possible we must determine its cost. The simplest compaction algorithm is to move all processes toward one end of the memory. All holes move in the other direction producing one large block of available memory. This scheme can be expensive.

Another possible solution to the external fragmentation is to permit the logical address space of the processes to be non-contiguous, thus allowing a process to be allocated physical memory wherever the latter is available.Two complementary techniques to achieve this solution are **paging** and**segmentation**. These techniques can be combined also. I will discuss these two schemes in the next article.

Source: http://www.go4expert.com/articles/memory-allocation-schemes-t22406/