BASICS OF MEMORY MANAGEMENT

What is Memory Management?

In a multiprogramming computer, the Operating System resides in a part of memory, and the rest is used by multiple processes. The task of subdividing the memory among different processes is called Memory Management. Memory management is a method in the operating system to manage operations between main memory and disk during process execution. The main aim of memory management is to achieve efficient utilization of memory.

Why Memory Management is Required?

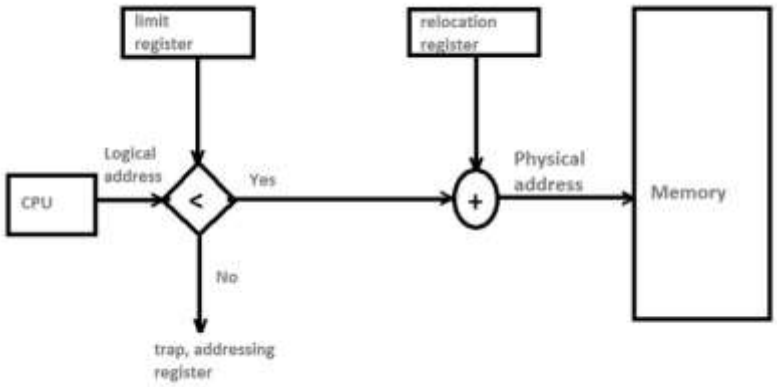
- Allocate and de-allocate memory before and after process execution.
- To keep track of used memory space by processes.
- To minimize fragmentation issues.
- To proper utilization of main memory.
- To maintain data integrity while executing of process.

Logical and Physical Address Space

- **Logical Address Space:** An address generated by the CPU is known as a "Logical Address". It is also known as a Virtual address. Logical address space can be defined as the size of the process. A logical address can be changed.
- **Physical Address Space:** An address seen by the memory unit (i.e the one loaded into the memory address register of the memory) is commonly known as a "Physical Address". A Physical address is also known as a Real address. The set of all physical addresses corresponding to these logical addresses is known as Physical address space. A physical address is computed by MMU. The run-time mapping from virtual to physical addresses is done by a hardware device Memory Management Unit(MMU). The physical address always remains constant.

Memory Protection

In Memory protection, we have to protect the operating system from user processes and which can be done by using a relocation register with a limit register. Here, the relocation register has the value of the smallest physical address whereas the limit register has the range of the logical addresses. These two registers have some conditions like each logical address must be less than the limit register. The memory management unit is used to translate the logical address with the value in the relocation register dynamically after which the translated (or mapped) address is then sent to memory.



In the above diagram, when the scheduler selects a process for the execution process, the dispatcher, on the other hand, is responsible for loading the relocation and limit registers with the correct values as part of the context switch as every address generated by the CPU is checked against these 2 registers, and we may protect the operating system, programs, and the data of the users from being altered by this running process.

Need of Memory protection:

Memory protection prevents a process from accessing unallocated memory in OS as it stops the software from seizing control of an excessive amount of memory and may cause damage that will impact other software which is currently being used or may create a loss of saved data. These resources of memory protection also help in detecting malicious or harmful applications, that may after damaged the processes of the operating system.

DYNAMIC LOADING AND DYNAMIC LINKING

Static and Dynamic Loading

Loading a process into the main memory is done by a loader. There are two different types of loading :

- **Static Loading:** Static Loading is basically loading the entire program into a fixed address. It requires more memory space.

- **Dynamic Loading:** The entire program and all data of a process must be in physical memory for the process to execute. So, the size of a process is limited to the size of physical memory. To gain proper memory utilization, dynamic loading is used. In dynamic loading, a routine is not loaded until it is called. All routines are residing on disk in a relocatable load format. One of the advantages of dynamic loading is that the unused routine is never loaded. This loading is useful when a large amount of code is needed to handle it efficiently.

Static and Dynamic Linking

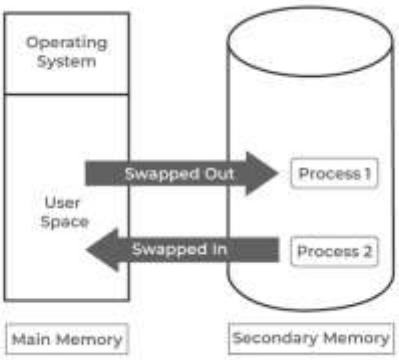
To perform a linking task a linker is used. A linker is a program that takes one or more object files generated by a compiler and combines them into a single executable file.

- **Static Linking:** In static linking, the linker combines all necessary program modules into a single executable program. So there is no runtime dependency. Some operating systems support only static linking, in which system language libraries are treated like any other object module.

- **Dynamic Linking:** The basic concept of dynamic linking is similar to dynamic loading. In dynamic linking, “Stub” is included for each appropriate library routine reference. A stub is a small piece of code. When the stub is executed, it checks whether the needed routine is already in memory or not. If not available then the program loads the routine into memory.

SWAPPING IN OS

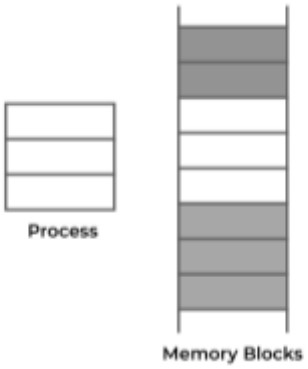
When a process is executed it must have resided in memory. Swapping is a process of swapping a process temporarily into a secondary memory from the main memory, which is fast compared to secondary memory. A swapping allows more processes to be run and can be fit into memory at one time. The main part of swapping is transferred time and the total time is directly proportional to the amount of memory swapped. Swapping is also known as roll-out, or roll because if a higher priority process arrives and wants service, the memory manager can swap out the lower priority process and then load and execute the higher priority process. After finishing higher priority work, the lower priority process swapped back in memory and continued to the execution process.



Swapping

CONTIGUOUS MEMORY ALLOCATION

The main memory should accommodate both the operating system and the different client processes. Therefore, the allocation of memory becomes an important task in the operating system. The memory is usually divided into two partitions: one for the resident operating system and one for the user processes. We normally need several user processes to reside in memory simultaneously. Therefore, we need to consider how to allocate available memory to the processes that are in the input queue waiting to be brought into memory. In adjacent memory allotment, each process is contained in a single contiguous segment of memory.



Contiguous Memory Allocation

Memory Allocation

To gain proper memory utilization, memory allocation must be allocated efficient manner. One of the simplest methods for allocating memory is to divide memory into several fixed-sized partitions and each partition contains exactly one process. Thus, the degree of multiprogramming is obtained by the number of partitions.

- **Multiple partition allocation:** In this method, a process is selected from the input queue and loaded into the free partition. When the process terminates, the partition becomes available for other processes.

- **Fixed partition allocation:** In this method, the operating system maintains a table that indicates which parts of memory are available and which are occupied by processes. Initially, all memory is available for user processes and is considered one large block of available memory. This available memory is known as a “Hole”. When the process arrives and needs memory, we search for a hole that is large enough to store this process. If the requirement is fulfilled then we allocate memory to process, otherwise keeping the rest available to satisfy future requests. While allocating a memory sometimes dynamic storage allocation problems occur, which concerns how to satisfy a request of size n from a list of free holes. There are some solutions to this problem:

Fragmentation

Fragmentation is defined as when the process is loaded and removed after execution from memory, it creates a small free hole. These holes cannot be assigned to new processes because holes are not combined or do not fulfil the memory requirement of the process. To achieve a degree of multiprogramming, we must reduce the waste of memory or fragmentation problems. In the operating systems two types of fragmentation:

1. **Internal fragmentation:** Internal fragmentation occurs when memory blocks are allocated to the process more than their requested size. Due to this some unused space is left over and creating an internal fragmentation problem.

   Example: Suppose there is a fixed partitioning used for memory allocation and the different sizes of blocks 3MB, 6MB, and 7MB space in memory. Now a new process p4 of size 2MB comes and demands a block of memory. It gets a memory block of 3MB but 1MB block of memory is a waste, and it cannot be allocated to other processes too. This is called internal fragmentation.

2. **External fragmentation:** In External Fragmentation, we have a free memory block, but we cannot assign it to a process because blocks are not contiguous.

   Example: Suppose (consider the above example) three processes p1, p2, and p3 come with sizes 2MB, 4MB, and 7MB respectively. Now they get memory blocks of size 3MB, 6MB, and 7MB allocated respectively. After allocating the process p1 process and the p2 process left 1MB and 2MB. Suppose a new process p4 comes and demands a 3MB block of memory, which is available, but we cannot assign it because free memory space is not contiguous. This is called external fragmentation.

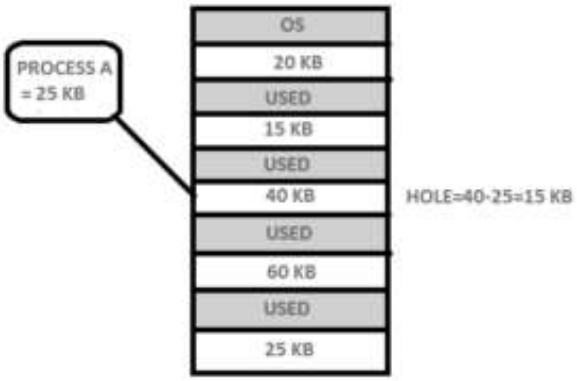
Both the first-fit and best-fit systems for memory allocation are affected by external fragmentation. To overcome the external fragmentation problem Compaction is used. In the compaction technique, all free memory space combines and makes one large block. So, this space can be used by other processes effectively.

Another possible solution to the external fragmentation is to allow the logical address space of the processes to be non-contiguous, thus permitting a process to be allocated physical memory wherever the latter is available.

PARTITION ALLOCATION ALGORITHMS

First Fit

In the First Fit, the first available free hole fulfil the requirement of the process allocated.

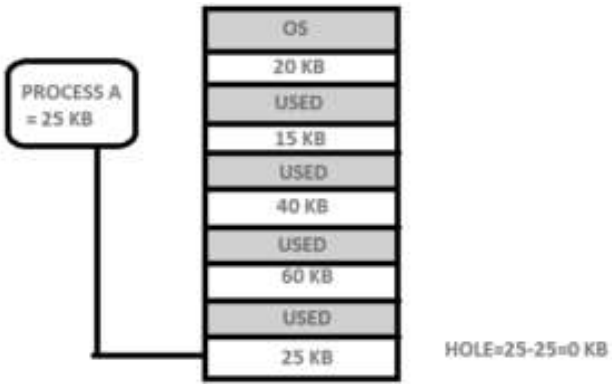


First Fit

Here, in this diagram, a 40 KB memory block is the first available free hole that can store process A (size of 25 KB), because the first two blocks did not have sufficient memory space.

Best Fit

In the Best fit, allocate the smallest hole that is big enough to process requirements. For this, we search the entire list, unless the list is ordered by size.



Best Fit

Here in this example, first, we traverse the complete list and find the last hole 25KB is the best suitable hole for Process A(size 25KB). In this method, memory utilization is maximum as compared to other memory allocation techniques.

Worst Fit

In the Worse Fit, allocate the largest available hole to process. This method produces the largest leftover hole.

Worst Fit

Here in this example, Process A (Size 25 KB) is allocated to the largest available memory block which is 60KB. Inefficient memory utilization is a major issue in the worst fit.

SEGMENTATION IN MEMORY MANAGEMENT

A process is divided into Segments. The chunks that a program is divided into which are not necessarily all of the exact sizes are called segments. Segmentation gives the

user's view of the process which paging does not provide. Here the user's view is mapped to physical memory.

Types of Segmentation in Operating Systems

- **Virtual Memory Segmentation:** Each process is divided into a number of segments, but the segmentation is not done all at once. This segmentation may or may not take place at the run time of the program.

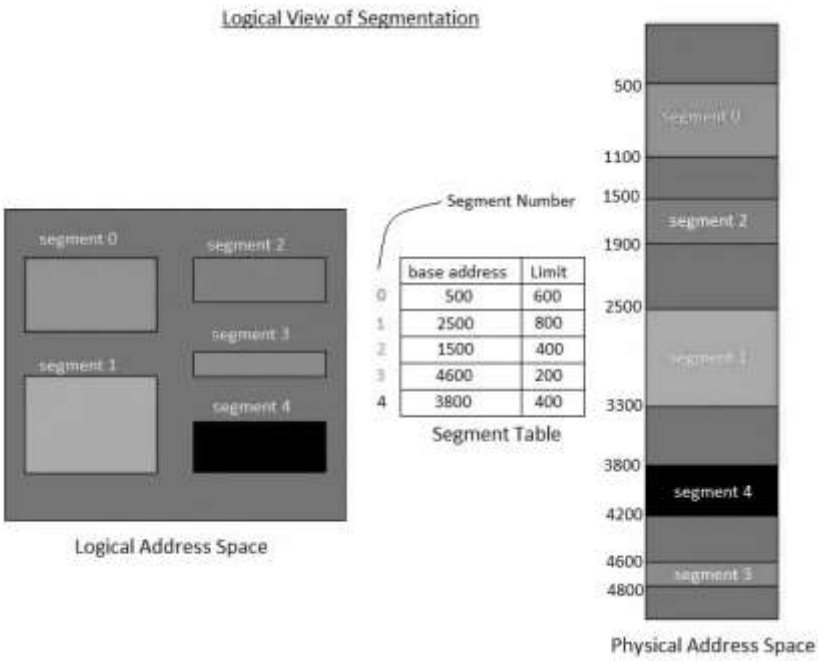
- **Simple Segmentation:** Each process is divided into a number of segments, all of which are loaded into memory at run time, though not necessarily contiguously.

There is no simple relationship between logical addresses and physical addresses in segmentation. A table stores the information about all such segments and is called Segment Table.

What is Segment Table?

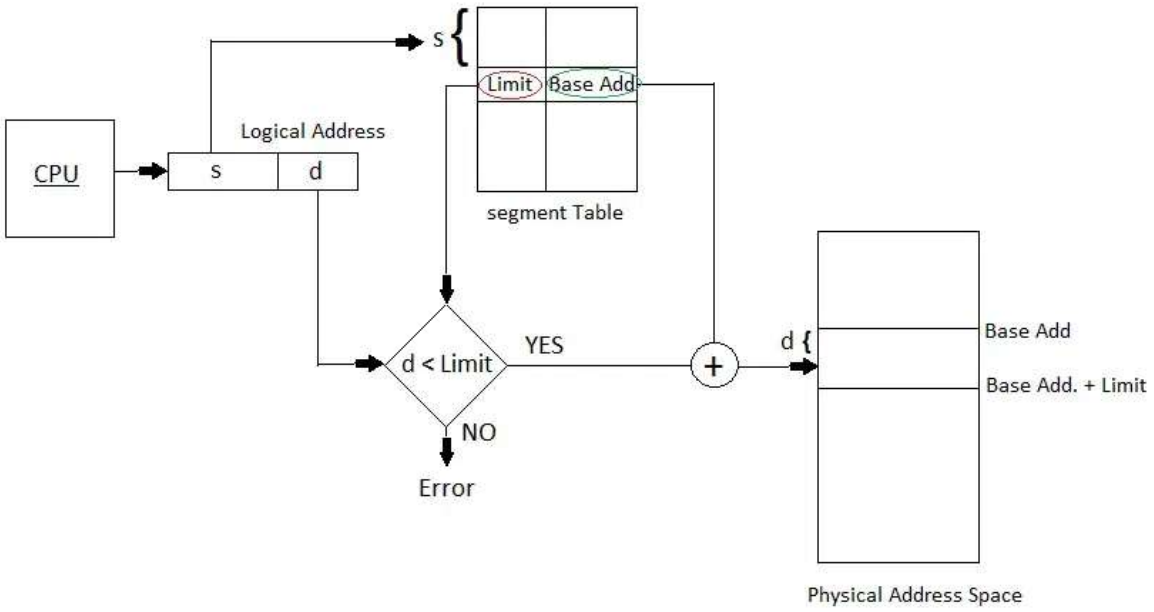
It maps a two-dimensional Logical address into a one-dimensional Physical address. It's each table entry has:

- **Base Address:** It contains the starting physical address where the segments reside in memory.
- **Segment Limit:** Also known as segment offset. It specifies the length of the segment.



Segmentation

Segmentation is crucial in efficient memory management within an operating system. For an in-depth understanding of memory management and other critical OS topics, explore the GATE CS Self-Paced Course, designed to help you succeed in the GATE exam. Translation of Two-dimensional Logical Address to Dimensional Physical Address.



Translation

The address generated by the CPU is divided into:

- **Segment number (s):** Number of bits required to represent the segment.
- **Segment offset (d):** Number of bits required to represent the position of data within a segment.

Advantages of Segmentation in Operating System

- **Reduced Internal Fragmentation** : Segmentation can reduce internal fragmentation compared to fixed-size paging, as segments can be sized according to the actual needs of a process. However, internal fragmentation can still occur if a segment is allocated more space than it is actually used.
- Segment Table consumes less space in comparison to Page table in paging.
- As a complete module is loaded all at once, segmentation improves CPU utilization.
- The user's perception of physical memory is quite similar to segmentation. Users can divide user programs into modules via segmentation. These modules are nothing more than separate processes' codes.
- The user specifies the segment size, whereas, in paging, the hardware determines the page size.
- Segmentation is a method that can be used to segregate data from security operations.
- **Flexibility:** Segmentation provides a higher degree of flexibility than paging. Segments can be of variable size, and processes can be designed to have multiple segments, allowing for more fine-grained memory allocation.
- **Sharing:** Segmentation allows for sharing of memory segments between processes. This can be useful for inter-process communication or for sharing code libraries.
- **Protection:** Segmentation provides a level of protection between segments, preventing one process from accessing or modifying another

process's memory segment. This can help increase the security and stability of the system.

Disadvantages of Segmentation in Operating System

- **External Fragmentation** : As processes are loaded and removed from memory, the free memory space is broken into little pieces, causing external fragmentation. This is a notable difference from paging, where external fragmentation is significantly lesser.
- Overhead is associated with keeping a segment table for each activity.
- Due to the need for two memory accesses, one for the segment table and the other for main memory, access time to retrieve the instruction increases.
- **Fragmentation:** As mentioned, segmentation can lead to external fragmentation as memory becomes divided into smaller segments. This can lead to wasted memory and decreased performance.
- **Overhead:** Using a segment table can increase overhead and reduce performance. Each segment table entry requires additional memory, and accessing the table to retrieve memory locations can increase the time needed for memory operations.
- **Complexity:** Segmentation can be more complex to implement and manage than paging. In particular, managing multiple segments per process can be challenging, and the potential for segmentation faults can increase as a result.

PAGING IN MEMORY MANAGEMENT

Paging is a memory management scheme that eliminates the need for a contiguous allocation of physical memory. This scheme permits the physical address space of a process to be non-contiguous.

- **Logical Address or Virtual Address (represented in bits):** An address generated by the CPU.

- **Logical Address Space or Virtual Address Space (represented in words or bytes):** The set of all logical addresses generated by a program.
- **Physical Address (represented in bits):** An address actually available on a memory unit.
- **Physical Address Space (represented in words or bytes):** The set of all physical addresses corresponding to the logical addresses.

Example:

- If Logical Address = 31 bits, then Logical Address Space = 231 words = 2 G words (1 G = 230)
- If Logical Address Space = 128 M words = 27 * 220 words, then Logical Address = log2 227 = 27 bits
- If Physical Address = 22 bits, then Physical Address Space = 222 words = 4 M words (1 M = 220)
- If Physical Address Space = 16 M words = 24 * 220 words, then Physical Address = log2 224 = 24 bits

The mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device and this mapping is known as the paging technique.

- The Physical Address Space is conceptually divided into several fixed-size blocks, called **frames**.
- The Logical Address Space is also split into fixed-size blocks, called **pages**.
- Page Size = Frame Size

Let us consider an example:

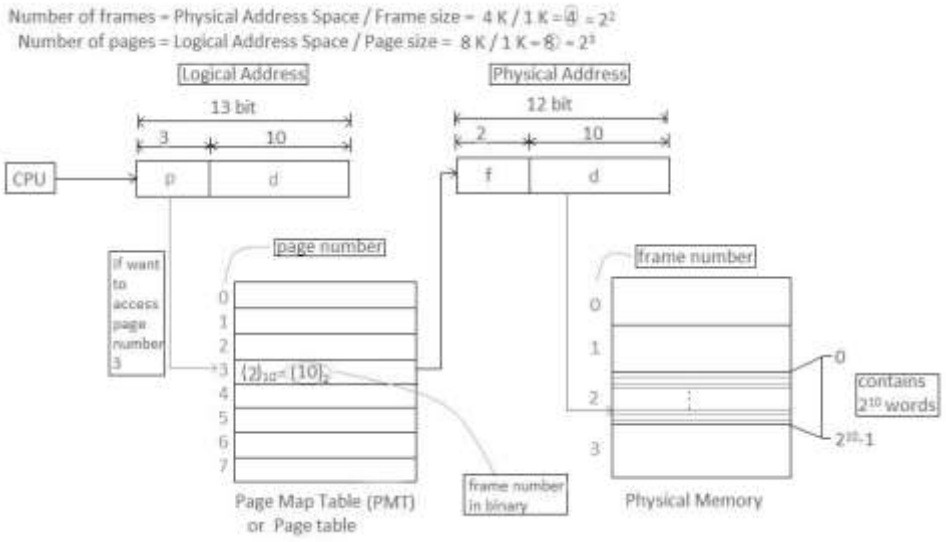
- Physical Address = 12 bits, then Physical Address Space = 4 K words
- Logical Address = 13 bits, then Logical Address Space = 8 K words
- Page size = frame size = 1 K words (assumption)

The address generated by the CPU is divided into:

- **Page Number(p):** Number of bits required to represent the pages in Logical Address Space or Page number
- **Page Offset(d):** Number of bits required to represent a particular word in a page or page size of Logical Address Space or word number of a page or page offset.

Physical Address is divided into:

- **Frame Number(f):** Number of bits required to represent the frame of Physical Address Space or Frame number frame
- **Frame Offset(d):** Number of bits required to represent a particular word in a frame or frame size of Physical Address Space or word number of a frame or frame offset.

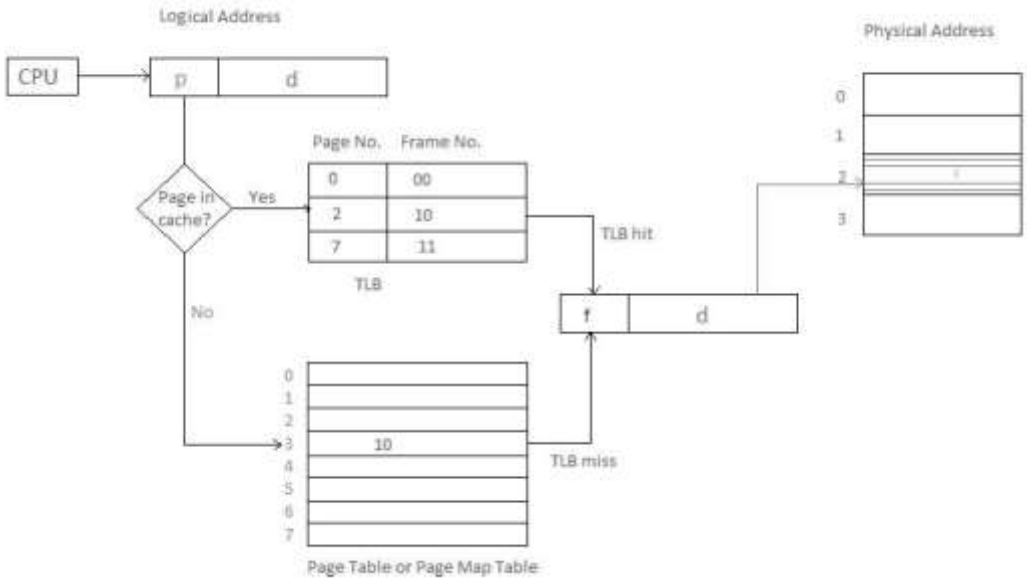


Paging

The hardware implementation of the page table can be done by using dedicated registers. But the usage of the register for the page table is satisfactory only if the page table is small. If the page table contains a large number of entries then we can use TLB(translation Look-aside buffer), a special, small, fast look-up hardware cache.

- The TLB is an associative, high-speed memory.
- Each entry in TLB consists of two parts: a tag and a value.

- When this memory is used, then an item is compared with all tags simultaneously. If the item is found, then the corresponding value is returned.



Page Map Table

Conclusion

This week we learned about the Basics of Memory Management, Dynamic Loading and Dynamic Linking, Swapping in OS, Contiguous Memory allocation, Partition Allocation Algorithms, Segmentation in Memory Management and Paging in Memory Management