OPERATING SYSTEM

ULEMU MPONELA
DMI ST JOHN THE BAPTIST UNIVERSITY : UNIT-III

Table of Contents

Unit III Memory Management:

Introduction- Address Binding - Dynamic Loading and Linking - Overlays - Logical and Physical Address Space – swapping - Contiguous Allocation - Internal & External Fragmentation. Non-Contiguous Allocation: Paging and Segmentation Schemes.

MEMORY MANAGEMENT

Memory management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution. Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free. It checks how much memory is to be allocated to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.

ADDRESS BINDING

The Address Binding refers to the mapping of computer instructions and data to physical memory locations. Both logical and physical addresses are used in computer memory. It assigns a physical memory region to a logical pointer by mapping a physical address to a logical address known as a virtual address. It is also a component of computer memory management that the OS performs on behalf of applications that require memory access.

Types of Address Binding in Operating System

There are mainly three types of an address binding in the OS. These are as follows:

1. Compile Time Address Binding
2. Load Time Address Binding
3. Execution Time or Dynamic Address Binding

Compile Time Address Binding

It is the first type of address binding. It occurs when the compiler is responsible for performing address binding, and the compiler interacts with the operating system to perform the address binding. In other words, when a program is executed, it allocates memory to the system code of the computer. The address binding assigns a logical address to the beginning of the memory segment to store the object code. Memory allocation is a long-term process and may only be modified by recompiling the program.

Load Time Address Binding

It is another type of address binding. It is done after loading the program in the memory, and it would be done by the operating system memory manager, i.e., loader. If memory allocation is specified when the program is assigned, no program in its compiled state may ever be transferred from one computer to another. Memory allocations in the executable code may already be in use by another program on the new system. In this case, the logical addresses of the program are not connected to physical addresses until it is applied and loaded into memory.

Execution Time or Dynamic Address Binding

Execution time address binding is the most popular type of binding for scripts that aren't compiled because it only applies to variables in the program. When a variable in a program is encountered during the processing of instructions in a script, the program seeks memory space for that variable. The memory would assign the space to that variable until the program sequence finished or unless a specific instruction within the script released the memory address connected to a variable.

DYNAMIC LOADING AND LINKING

Dynamic Loading

The dynamic binding is a method to obtain better memory-space utilization. A routine is not loaded until it is called, until then all routine remains in the disk as relocatable code format

A routing can call another routine, but it checks if another routine is loaded or not. If not, then relocatable linking loader is called to load the routine into memory and update program's address tables to reflect this change. The control is passed to the newly loaded routine.

The total program may be large but only a small portion is loaded. The unused routine is never loaded into memory.

It is the programmer's responsibility to implement dynamic loading, the OS can provide library routines to implement the dynamic loading.

Dynamic Linking

In static linking, the system libraries are treated like any other object modules and combined by the loader into the executing program.

The dynamic linking, similar to dynamic loading is delayed until run-time. This feature is used with system libraries such as language subroutine libraries. Without this facility, each program must include a copy of its language library in the executable image which wastes memory space

There are several benefits of dynamic linking:

- library routines not part of program code. A program is much smaller without the routines and loads faster into the memory.
- One copy of library routing is referred by many programs, therefore, saving disk space and memory space.
- Fixing bugs is easy because you are only required to fix a single routine, rather than individual copies of it.

The library can be updated with a newer version. Since the program must use the correct version of the library, the version information is included with the program

and the library. Also, several version of the library is loaded into the memory, so that programs could use the correct copy of the library

If the library has minor changes it retains the same version number. The version is incremented if the library has major changes, and affects the only program that is compiled using a newer version of the library. Old programs use an older version of libraries. This system is known as shared libraries.

Libraries require help from the operating system. If the processes in memory are protected from one another, OS checks if the required routine is already linked to another process, or allow multiple processes to access same memory addresses.

OVERLAYS

The main problem in Fixed partitioning is the size of a process has to be limited by the maximum size of the partition, which means a process can never be span over another. In order to solve this problem, earlier people have used some solution which is called as Overlays.

The concept of overlays is that whenever a process is running it will not use the complete program at the same time, it will use only some part of it. Then overlays concept says that whatever part you required, you load it and once the part is done, then you just unload it, means just pull it back and get the new part you required and run it.
Formally, the process of transferring a block of program code or other data into internal memory, replacing what is already stored.

Sometimes it happens that compare to the size of the biggest partition, the size of the program will be even more, then, in that case, you should go with overlays.

So overlay is a technique to run a program that is bigger than the size of the physical memory by keeping only those instructions and data that are needed at any given time. Divide the program into modules in such a way that not all modules need to be in the memory at the same time.

Advantage

- Reduce memory requirement
- Reduce time requirement

Disadvantage

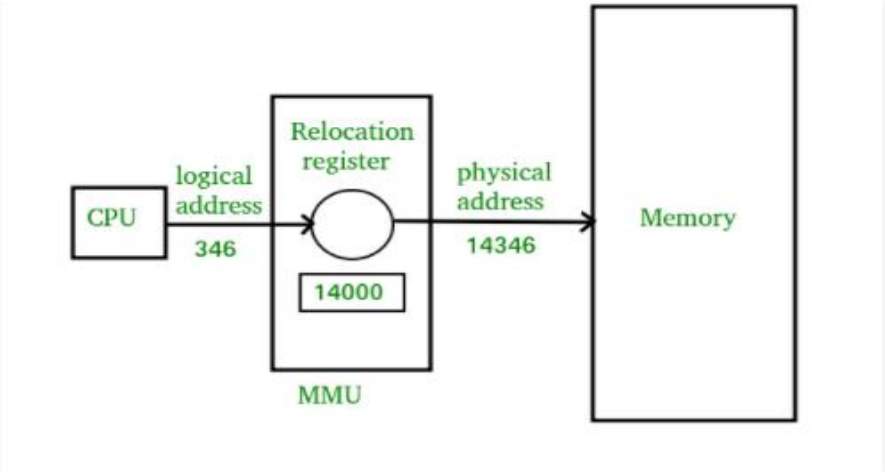
- Overlap map must be specified by programmer
- Programmer must know memory requirement
- Overlapped module must be completely disjoint
- Programming design of overlays structure is complex and not possible in all cases

LOGICAL AND PHYSICAL ADDRESSES

Logical Address is generated by CPU while a program is running. The logical address is virtual address as it does not exist physically, therefore, it is also known as Virtual Address. This address is used as a reference to access the physical memory location by

CPU. The term Logical Address Space is used for the set of all logical addresses generated by a program's perspective.
The hardware device called Memory-Management Unit is used for mapping logical address to its corresponding physical address.

Physical Address identifies a physical location of required data in a memory. The user never directly deals with the physical address but can access by its corresponding logical address. The user program generates the logical address and thinks that the program is running in this logical address but the program needs physical memory for its execution, therefore, the logical address must be mapped to the physical address by MMU before they are used. The term Physical Address Space is used for all physical addresses corresponding to the logical addresses in a Logical address space.



Difference Between Logical and Physical Addresses

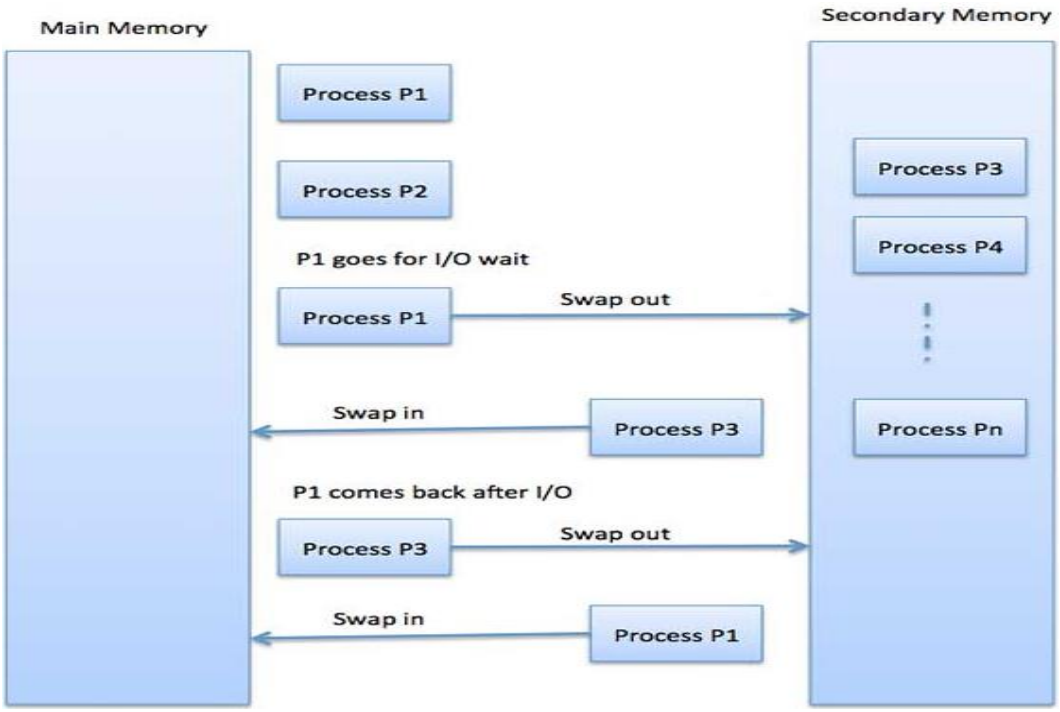
1. The basic difference between Logical and physical address is that Logical address is generated by CPU in perspective of a program whereas the physical address is a location that exists in the memory unit.
2. Logical Address Space is the set of all logical addresses generated by CPU for a program whereas the set of all physical address mapped to corresponding logical addresses is called Physical Address Space.
3. The logical address does not exist physically in the memory whereas physical address is a location in the memory that can be accessed physically.
4. Identical logical addresses are generated by Compile-time and Load time address binding methods whereas they differ from each other in run-time address binding method.
5. The logical address is generated by the CPU while the program is running whereas the physical address is computed by the Memory Management Unit (MMU).

Parameter	LOGICAL ADDRESS	PHYSICAL ADDRESS
Basic	generated by CPU	location in a memory unit
Address Space	Logical Address Space is set of all logical addresses generated by CPU in reference to a program.	Physical Address is set of all physical addresses mapped to the corresponding logical addresses.
Visibility	User can view the logical address of a program.	User can never view physical address of program.
Generation	generated by the CPU	Computed by MMU
Access	The user can use the logical address to access the physical address.	The user can indirectly access physical address but not directly.

SWAPPING

Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.

Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel and that's the reason Swapping is also known as a technique for memory compaction.



The total time taken by swapping process includes the time it takes to move the entire process to a secondary disk and then to copy the process back to memory, as well as the time the process takes to regain main memory.

Let us assume that the user process is of size 2048KB and on a standard hard disk where swapping will take place has a data transfer rate around 1 MB per second. The actual transfer of the 1000K process to or from memory will take.

```
2048KB / 1024KB per second
= 2 seconds
= 2000 milliseconds
```

CONTIGOUS MEMORY ALLOCATION

A contiguous memory allocation is a memory management technique where whenever there is a request by the user process for the memory, a single section of the contiguous memory block is given to that process according to its requirement.

It is achieved by dividing the memory into fixed-sized partitions or variable-sized partitions.

Fixed-sized partition scheme:

It is also known as Static-partitioning.
The system is divided into fixed-sized partitions.

In this scheme, each partition may contain exactly one process. This process limits the extent of multiprogramming, as the number of partitions decides the number of processes.

Variable-sized partition scheme:

It is also known as Dynamic partitioning. In this, the scheme allocation is done dynamically

The size of each partition is not declared initially, and only once we know the size of the process. The size of the partition and the process is equal, hence preventing internal fragmentation.

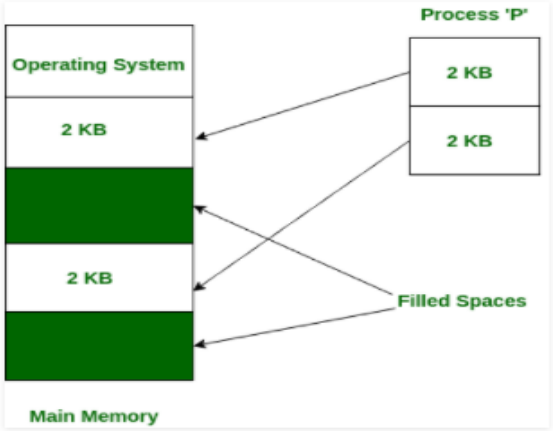
When the process is smaller than the partition, some size of the partition gets wasted, this is known as internal fragmentation. It is a concern in static partitioning, but dynamic partitioning aims to solve this issue.

NON-CONTIGOUS MEMORY ALLOCATION

In non-contiguous allocation, operating system needs to maintain the table which is called Page Table for each process which contains the base address of each block which is acquired by the process in memory space. In non-contiguous memory allocation, different parts of a process is allocated different places in Main Memory. Spanning is allowed which is not possible in other techniques like Dynamic or Static

Contiguous memory allocation. That's why paging is needed to ensure effective memory allocation. Paging is done to remove External Fragmentation.

Working:
Here a process can be spanned across different spaces in main memory in non-consecutive manner. Suppose process P of size 4KB. Consider main memory have two empty slots each of size 2KB. Hence total free space is, 2*2= 4 KB. In contiguous memory allocation, process P cannot be accommodated as spanning is not allowed. In contiguous allocation, space in memory should be allocated to whole process. If not, then that space remains unallocated. But in Non-Contiguous allocation, process can be divided into different parts and hence filling the space in main memory. In this example, process P can be divided into two parts of equal size – 2KB. Hence one part of process P can be allocated to first 2KB space of main memory and other part of process P can be allocated to second 2KB space of main memory. Below diagram will explain in better way:



EXTERNAL AND INTERNAL FRAGMENTATION

Fragmentation is an unwanted problem in the operating system in which the processes are loaded and unloaded from memory, and free memory space is fragmented. Processes can't be assigned to memory blocks due to their small size, and the memory blocks stay unused. It is also necessary to understand that as programs are loaded and deleted from memory, they generate free space or a hole in the memory. These small blocks cannot be allotted to new arriving processes, resulting in inefficient memory use.

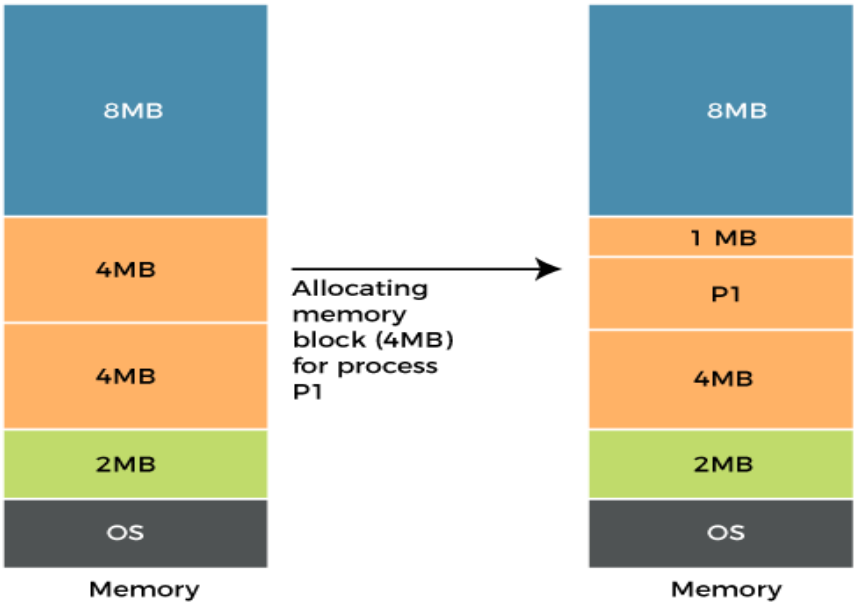
The conditions of fragmentation depend on the memory allocation system. As the process is loaded and unloaded from memory, these areas are fragmented into small pieces of memory that cannot be allocated to incoming processes. It is called fragmentation.

Internal Fragmentation

When a process is allocated to a memory block, and if the process is smaller than the amount of memory requested, a free space is created in the given memory block. Due to this, the free space of the memory block is unused, which causes internal fragmentation.

For Example

Assume that memory allocation in RAM is done using fixed partitioning (i.e., memory blocks of fixed sizes). **2MB, 4MB, 4MB**, and **8MB** are the available sizes. The Operating System uses a part of this RAM.



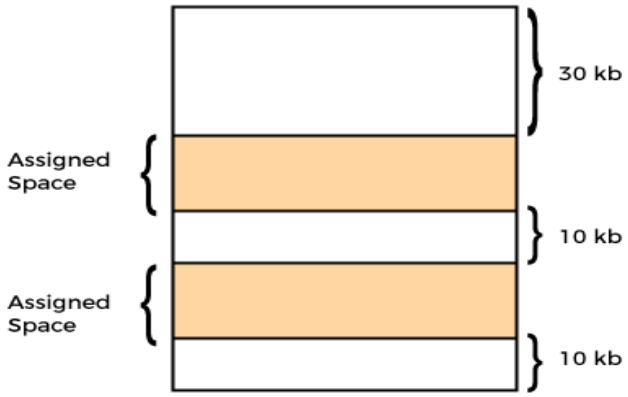
Let's suppose a process **P1** with a size of **3MB** arrives and is given a memory block of **4MB**. As a result, the **1MB** of free space in this block is unused and cannot be used to allocate memory to another process. It is known as internal fragmentation.

The problem of internal fragmentation may arise due to the fixed sizes of the memory blocks. It may be solved by assigning space to the process via dynamic partitioning. Dynamic partitioning allocates only the amount of space requested by the process. As a result, there is no internal fragmentation.

External Fragmentation

External fragmentation happens when a dynamic memory allocation method allocates some memory but leaves a small amount of memory unusable. The quantity of available memory is substantially reduced if there is too much external fragmentation. There is enough memory space to complete a request, but it is not contiguous. It's known as external fragmentation.

For Example



Process 05 needs 45kb memory space

Let's take the example of external fragmentation. In the above diagram, you can see that there is sufficient space **(50 KB)** to run a process **(05) (need 45KB)**, but the memory is not contiguous. You can use compaction, paging, and segmentation to use the free space to execute a process.

This problem occurs when you allocate RAM to processes continuously. It is done in paging and segmentation, where memory is allocated to processes non-contiguously. As a result, if you remove this condition, external fragmentation may be decreased.

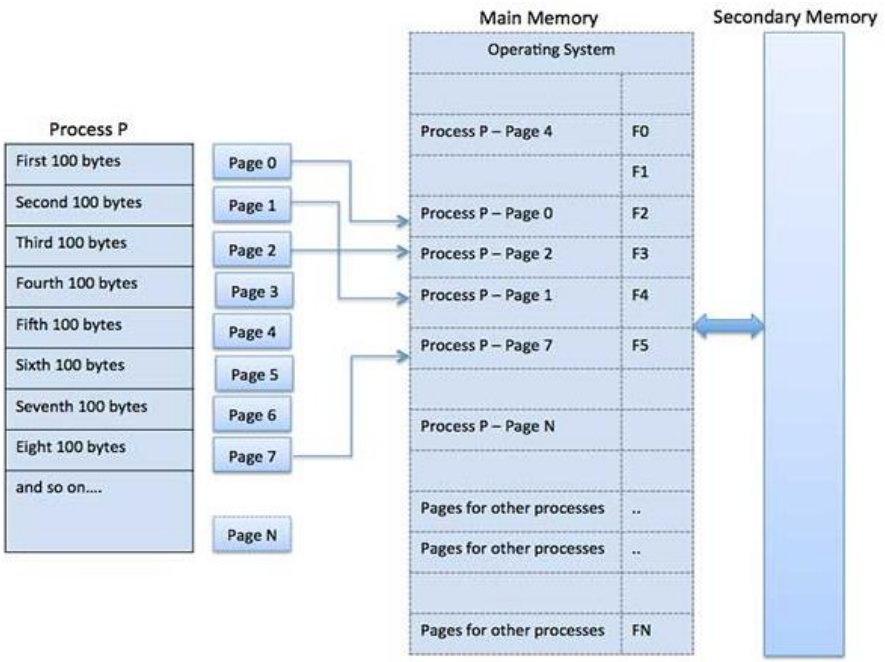
Compaction is another method for removing external fragmentation. External fragmentation may be decreased when dynamic partitioning is used for memory allocation by combining all free memory into a single large block. The larger memory block is used to allocate space based on the requirements of the new processes. This method is also known as defragmentation.

PAGING

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard that's set up to emulate the computer's RAM. Paging technique plays an important role in implementing virtual memory.

Paging is a memory management technique in which process address space is broken into blocks of the same size called pages (size is power of 2, between 512 bytes and 8192 bytes). The size of the process is measured in the number of pages.

Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.



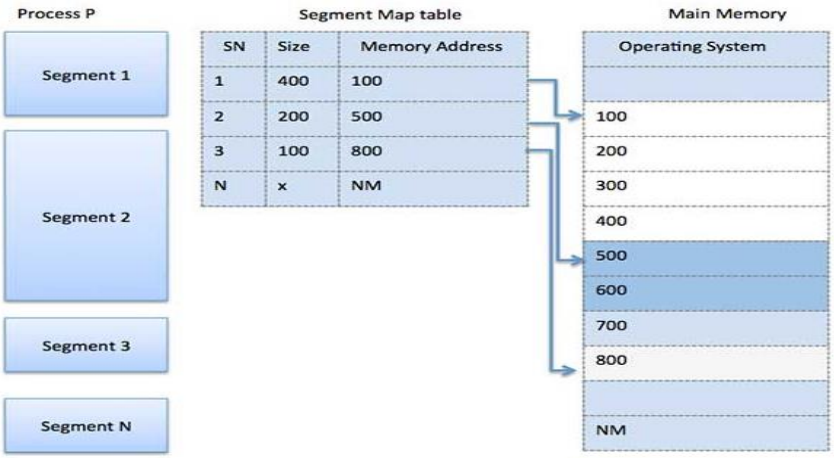
SEGMENTATION

Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions. Each segment is actually a different logical address space of the program.

When a process is to be executed, its corresponding segmentation are loaded into non-contiguous memory though every segment is loaded into a contiguous block of available memory.

Segmentation memory management works very similar to paging but here segments are of variable-length where as in paging pages are of fixed size.

A program segment contains the program's main function, utility functions, data structures, and so on. The operating system maintains a segment map table for every process and a list of free memory blocks along with segment numbers, their size and corresponding memory locations in main memory. For each segment, the table stores the starting address of the segment and the length of the segment. A reference to a memory location includes a value that identifies a segment and an offset.


Process P
Segment 1
Segment 2
Segment 3
Segment N
Segment Map table
SN
Size
Memory Address
1
400
100
2
200
500
3
100
800
N
x
NM
Main Memory
Operating System
100
200
300
400
500
600
700
800
NM