Introduction to Memory Management

- The memory management modules of an operating system are concerned with the management of primary memory. In a multiprogramming system, the user part of memory must be further subdivided to accommodate multiple processes. The task of subdivision is carried out dynamically by the operating system and is known as memory management.
- Memory consists of a large array of words or bytes each with its own address. Memory that the processors directly access for instructions and data.

4.1 Memory Management Functions

1. Keeping track of the status of each memory location. Whether the memory location is allocated or free.

2. Determining allocation policy for memory.

3. Memory allocation technique, memory allocation information must be updated.

4. Deallocation technique and policy. A process may release the allocated memory.

Central processing unit fetches instructions from memory according to the value of program counter. These instructions may cause additional loading from and storing to specific memory addresses. In this section, we discuss the memory related parameters like address binding, physical address, logical address, and dynamic loading etc.

Logical Vs Physical Address Space

The address generated by the CPU is a logical address, whereas the address actually seen by the memory hardware is a physical address.

Addresses bound at compile time or load time have identical logical and physical addresses.

Addresses created at execution time, however, have different logical and physical addresses.

In this case the logical address is also known as a virtual address, and the two terms are used interchangeably by our text.

The set of all logical addresses used by a program composes the logical address space, and the set of all corresponding physical addresses composes the physical address space.

The run time mapping of logical to physical addresses is handled by the memory-management unit, MMU.

The MMU can take on many forms. One of the simplest is a modification of the base-register scheme described earlier.

The base register is now termed a relocation register, whose value is added to every memory request at the hardware level.

Note that user programs never see physical addresses. User programs work entirely in logical address space, and any memory references or manipulations are done using purely logical addresses. Only when the address gets sent to the physical memory chips is the physical memory address generated.

Figure below shows the dynamic relocation. Dynamic relocation implies that mapping from the virtual address space to the physical address space is performed at run-time, usually with some hardware assistance. MS-DOS operating system running on the Intel 80x86 family of processors use four relocation registers when loading and running processes.

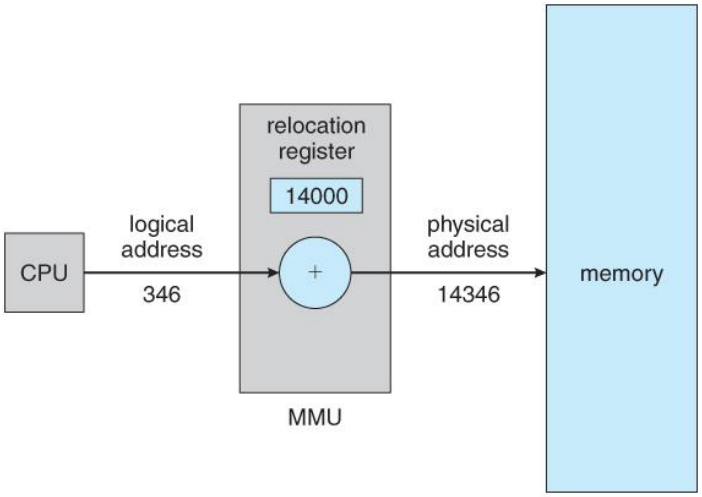


Figure Dynamic relocation using a relocation register

Relocation is performed by hardware and is invisible to the user. Dynamic relocation makes it possible to move a partially executed process from one area of memory into another without affecting.

Memory Protection

The system shown in Figure below allows protection against user programs accessing areas that they should not, allows programs to be relocated to different memory starting addresses as needed, and allows the memory space devoted to the OS to grow or shrink dynamically as needs change.

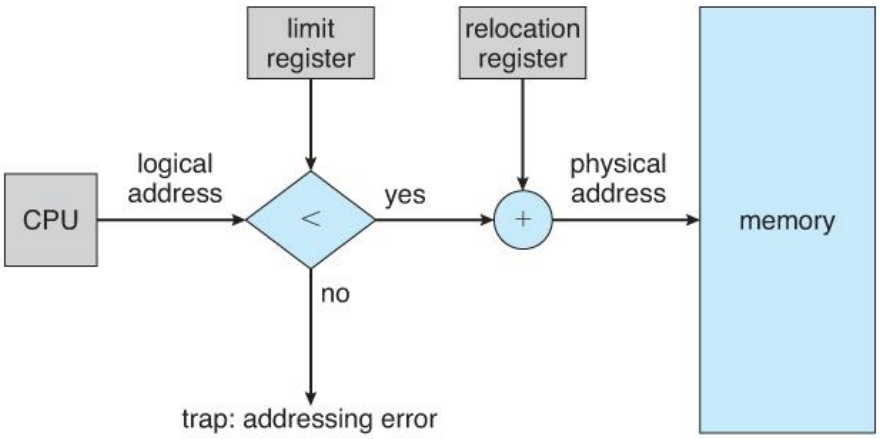


Figure Hardware support for relocation and limit registers

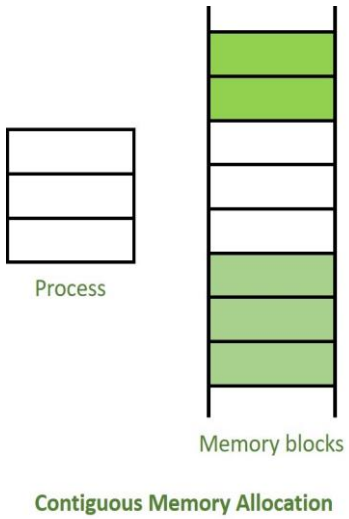
4.2 Contiguous Memory Allocation

Contiguous memory allocation is basically a method in which a single contiguous section/part of memory is allocated to a process or file needing it. Because of this all the available memory space resides at the same place together, which means that the freely/unused available memory partitions are not distributed in a random fashion here and there across the whole memory space.

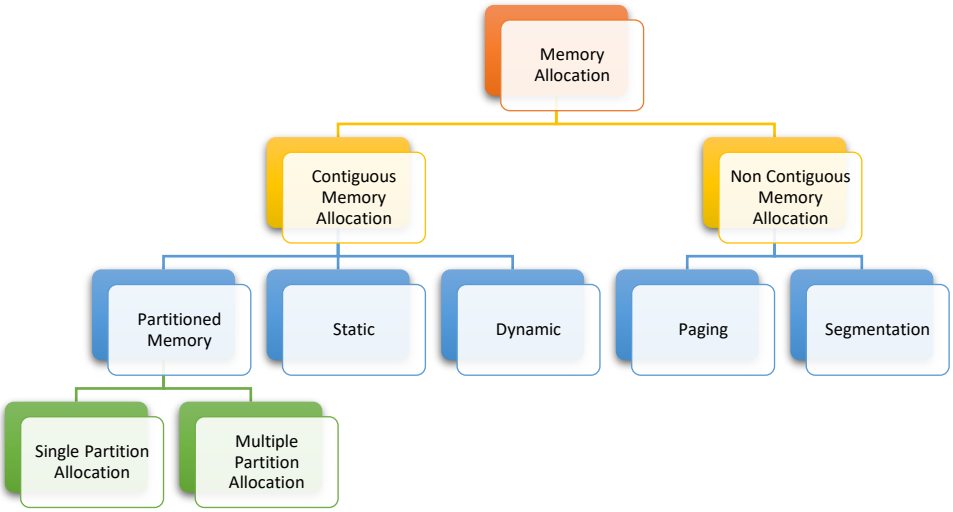
The main memory is a combination of two main portions

- one for the operating system and
- other for the user program.

We can implement/achieve contiguous memory allocation by dividing the memory partitions into fixed size partitions



Contiguous Memory Allocation



1 Partitioned Memory

1.1 Single-Partition Allocation

If the operating system is residing in low memory, and the user processes are executing in high memory, and operating-system code and data are protected from changes by the user processes. We also need protect the user processes from one another. We can provide these 2 protections by using a relocation register.

The relocation register contains the value of the smallest physical address; the limit register contains the range of logical addresses (for example, relocation = 100,040 and limit = 74,600). With relocation and limit registers, each logical address must be less than the limit register; the MMU maps the logical address dynamically by adding the value in the relocation register. This mapped address is sent to memory.

The relocation-register scheme provides an effective way to allow the operating system size to change dynamically.

1.2 Multiple-Partition Allocation

One of the simplest schemes for memory allocation is to divide memory into a number of fixed-sized partitions. Each partition may contain exactly one process. Thus, the degree of multiprogramming is bound by the number of partitions. 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.

The operating system keeps a table indicating which parts of memory are available and which are occupied. Initially, all memory is available for user processes, and is considered as one large block, of available memory, a hole. When a process arrives and needs memory, we search for a hole large enough for this process.

For example, assume that we have 2560K of memory available and a resident operating system of 400K. This situation leaves 2160K for user processes. FCFS job scheduling, we can immediately allocate memory to processes P1, P2, P3. Hole's size 260K that cannot be used by any of the remaining processes in the input queue. Using a round-robin CPU-scheduling with a quantum of 1 time unit, process will terminate at time 14, releasing its memory.

2 Multiprogramming with Fixed (Static) Partitions

This method allows multiple processes to execute simultaneously.

Here, memory is shared among operating system and various simultaneously running processes.

Multiprogramming increases the CPU utilization. CPU can be kept busy almost all the time by keeping more than one process simultaneously in the memory.

Here, memory is divided into fixed size partitions. Size can be equal or unequal for different partitions. Generally, unequally sized partitions are used for better utilization of memory.

Each partition can accommodate exactly one process. Means, only single process can be placed in one partition.

Whenever any program needs to be loaded in memory, a free partition big enough to hold the program is found. This partition will be allocated to that program (or process).

If there is no free partition available of required size, then that process needs to wait. Such process will be put in a queue.

There are two possible ways to implement this method with queues as given below:

1. Using Multiple Input Queue

2. Using Single Input Queue

Figure depicts both of these two possible implementations.

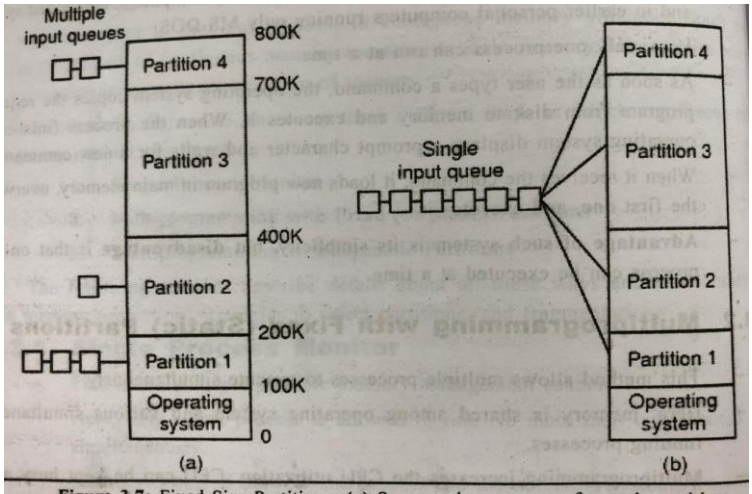


Figure 3.7: Fixed Size Partitions: **(a)** Separate input queues for each partition, **(b)** Single input queue

1. Using Multiple Input Queues:

Here, for each partition, separate queue is maintained. Whenever any process comes, it is put in a queue for the smallest partition large enough to hold it. Whenever any partition becomes free, a process from the front end of the queue is allocated that partition.

Disadvantage of this method: if the queue for a large partition is empty, but fa a small partition is full, small process needs to wait to get memory. Here, free memory is available in large partition-

n, but it cannot be used for small process. This case is given for partition 1 and 3. Partition 3 is empty, but it cannot be used for processes of partition 1.

2. Using Single Input Queue:

Here, only single queue is maintained for all the partitions.

Whenever any partition becomes free, a process from queue is selected, which can be hold by that partition. And, that partition is allocated for that process.

A process can be selected in two ways:

First, just select the process which is near to the front of the queue. **Disadvantage** here is large partition will be wasted on small process.

Secondly, Search the entire queue and select the largest Process which can fit in that partition. Disadvantage here is, starvation is possible for small processes.

Advantages of Static Partitions:

1) Implementation is simple.
2) Processing overheads are low.

Disadvantages of Static Partitions

1) Degree of multi programming is fixed here. The number of maximum possible processes that can be executed simultaneously cannot be changed.
2) Here, partitions are of fixed size. So, any space in a partition not used by process is wasted. This is called Internal Fragmentation.

3 Multiprogramming with Dynamic Partitions

This method also allows multiple processes to execute simultaneously.

Here, memory is shared among operating system and various simultaneously running processes.

Here, memory is not divided into any fixed size partitions. Also the number of partitions is not fixed. Process is allocated exactly as much memory as it requires.

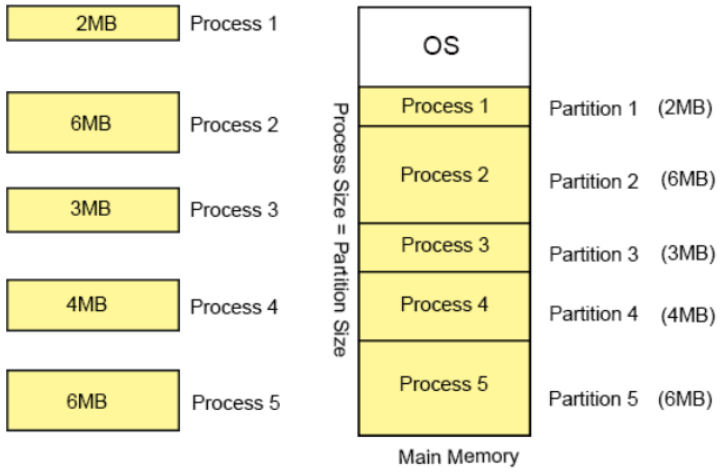
Initially, the entire available memory is treated as a single free partition.

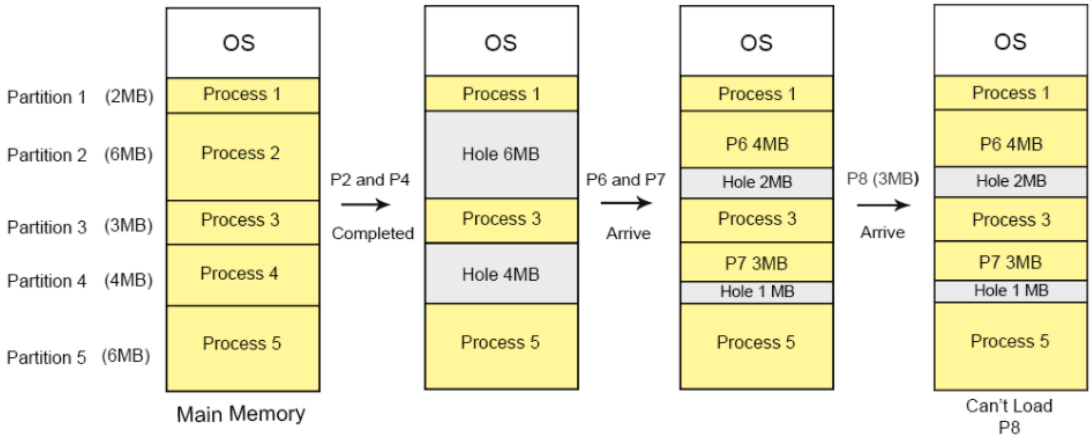
Whenever any process enters in a system, a chunk of free memory big enough to fit the process is found and allocated. The remaining unoccupied space is treated as another free partition.

If enough free memory is not available to fit the process, process needs to wait until required memory becomes available.

Whenever any process gets terminate, it releases the space occupied. If the released free space is contiguous to another free partition, both the free partitions are clubbed together into a single free partition.

The following figure below explains the working of this method. Initially, only an operating system has been loaded in memory as shown in figure. Then after, whenever new process comes, it will be allocated memory. In a similar way, whenever any process terminates, memory will be released.





Advantages of Dynamic Partitions:

i) Better utilization of memory. There is no internal fragmentation.

ii) Degree of multiprogramming is not fixed here.

Disadvantages of Dynamic Partitions:

i) External fragmentation:

Memory is allocated when process enters in the system and released when it terminates. These operations eventually result in small holes in the memory.

These holes will be so small that no processes can be loaded in it. But total size of all holes may be big enough to hold any process.

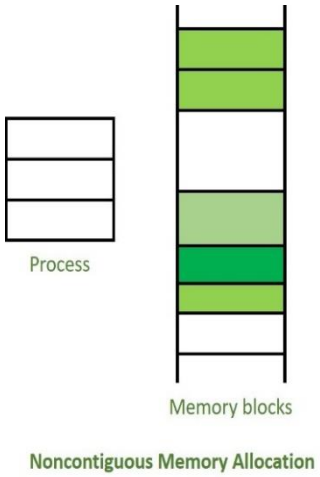
But, as memory is allocated contiguously here, these holes can't be used. This type of memory wastage is called external fragmentation.

Solution to this problem is compaction or de-fragmentation of the memory.

4.3 Non-Contiguous Memory Allocation

Non-Contiguous memory allocation is basically a method on the contrary to contiguous allocation method, allocates the memory space present in different locations to the process as per it's requirements. As all the available memory space is in a distributed pattern so the freely available memory space is also scattered here and there.

This technique of memory allocation helps to reduce the wastage of memory, which eventually gives rise to Internal and external fragmentation.



Noncontiguous Memory Allocation

4.3.1 Non-contiguous Memory Allocation: Paging

Most operating systems use Non-Contiguous Memory Allocation method for allocating memory. In this method, logical address space of the process is divided into partitions. For each partition contiguous chunk of free memory is allocated in physical memory.

Physical address space of a process will not be contiguous now. Physical addresses Will be scattered over entire memory.

There are two kinds of allocation techniques:

1. Paging
2. Segmentation

Basic Concept

The logical address space of a process is divided into blocks of fixed size, called Pages.

Also, the physical memory is divided into blocks of fixed size, called Frames (or frames).

The pages and frames will be of the same size. This size will be typically a power of 2.It varies between 512 bytes to a few MB.

Whenever a process is to be executed, its pages are moved from secondary storage, i.e. a disk. to the available frames in physical memory. So, memory allocation refers to the task of finding free frames in memory, allocating them to pages. and keeping track of which frame belongs to which page.

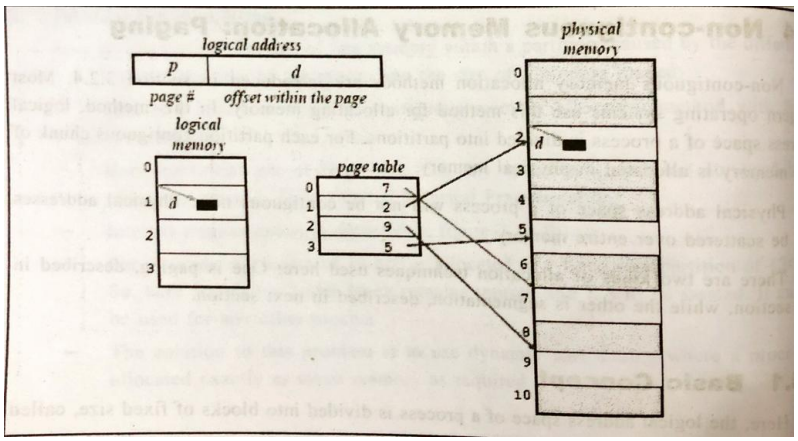
Operating system maintains a table, called **page table**, for each process.

The page table is indexed by a page number. The information about frame numbers, in which pages are stored, is kept in this page table.

Here, logical address is divided into two parts: **i) page number** which gives the number of a page; and, **ii) an offset,** which gives the actual location within a page.

The following figure 3.12 gives idea about this method.

Here' the logical address space of a process, also called **logical memory** has been into four pages. Physical memory contains total of 11 frames. Also, a page table is given which shows the mapping between pages and frames.



Implementation

The figure above depicts how the paging scheme is implemented. It describes how logical addresses are converted to physical addresses.

A table, called page table, is used to implement paging. When a process is to execute, its pages are moved to free frames in physical memory. The information about frame number, where a page is stored, is kept in this page table.

During the process execution, a CPU generates a logical address (L) to access instruction or data from a particular location. This logical address is divided into two parts: a number (p) and an offset (d) within a page.

Logical Address (L):

Page number (p)	Offset (d)

The page number is used to search the page table. Corresponding frame number (f) obtained from the page table. This frame number indicates the actual frame on physical memory in which the page is stored.

The physical address (P) of a particular location is obtained by combining the number (f) with the offset (d).

Physical Address (P):

Frame number (f)	Offset (d)

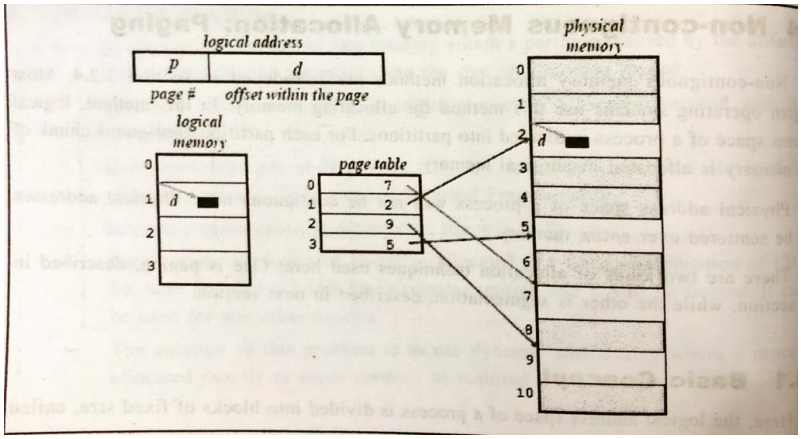


Figure: Address Translation in Paging

Address Translation:

- The basic idea behind paging is to divide physical memory into a number of equal sized blocks called ***frames***, and to divide a programs logical memory space into blocks of the same size called ***pages.***
- Any page (from any process) can be placed into any available frame.
- The ***page table*** is used to look up what frame a particular page is stored in at the moment. In the following example, for instance, page 2 of the program's logical memory is currently stored in frame 3 of physical memory:

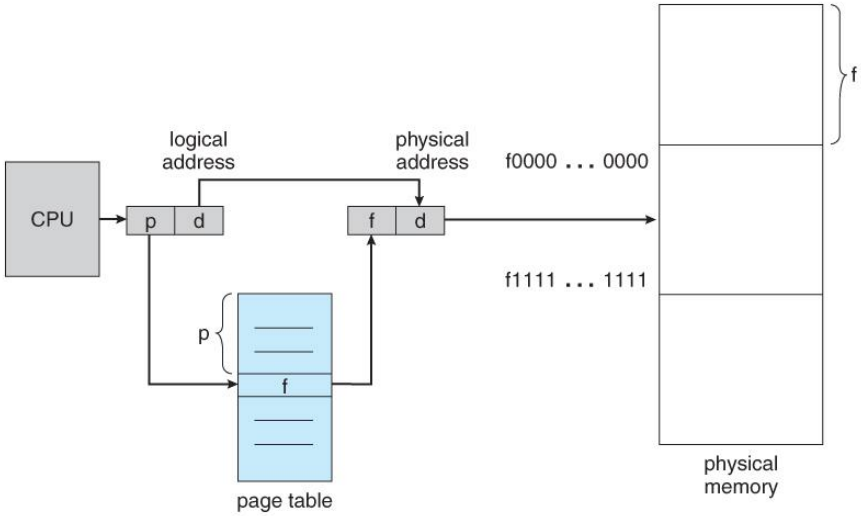


Figure Paging hardware

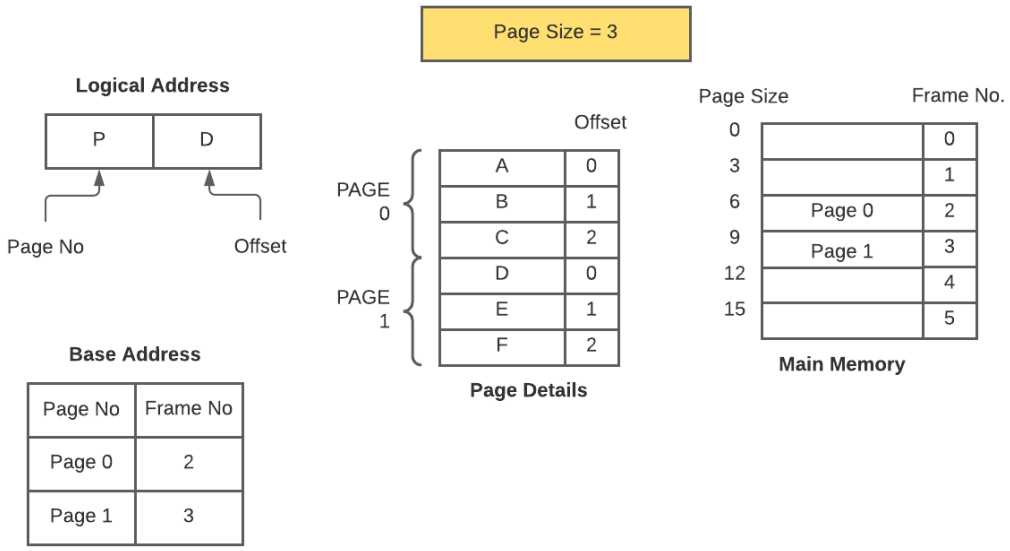


Figure for Address Translation

Here the page size is 3.

Logical Address is divided into 2 parts:

1) Page No
2) Offset

The Physical Address formula is as below:

Physical Address = Base Address * Page Size + offset of logical address

Question 1: Calculate Physical Address for the instruction D for page size 3

Base Address = 3

Page Size = 3

Offset of D = 0

Physical Address = 3*3+0 = **9**

Question 2: Calculate Physical Address for the instruction C for page size 3

Base Address = 2

Page Size = 3

Offset of C = 2

Physical Address = 2*3+2 = **8**

Advantages - Disadvantages of Basic Paging Method

Advantages:

1. Allocation and de-allocation of memory is extremely fast. For this, only the list free frames need to be maintained. No need of any algorithms such as first best fit, etc.

2. Swap-in and Swap-out operations are fast. As page size matches disk block size data can be moved to and from disk very easily.

3. There is no external fragmentation. Each and every frame from memory can allocated to processes.

Disadvantages:

1. Additional memory reference is required to read information from page table. Every instruction or data reference required two memory accesses: One for page table, and one for instruction or data.

2. Size of page table may be too large to keep it in main memory. Page table contains entry for all the pages in logical address space. For large process, page table be large.

3. Internal fragmentation: A process size may not be an exact multiple of the page size. So, some space would remain unoccupied in the last page of a process. The results in internal fragmentation.

Variations over Basic Method

The two most significant problems with the basic paging scheme are: high access time and large size of a page table.

Most modern operating systems overcome these two problems by using one or variations over the basic paging scheme. Some of the most widely used variations are.

1. Translation Look-aside Buffer (TLB)

2. Multi Level Page Tables

3. Inverted Page Table (IPT)

4. Hashed Page Table(HPT)

These variations are explained below:

1. Translation Look-aside Buffer (TLB)

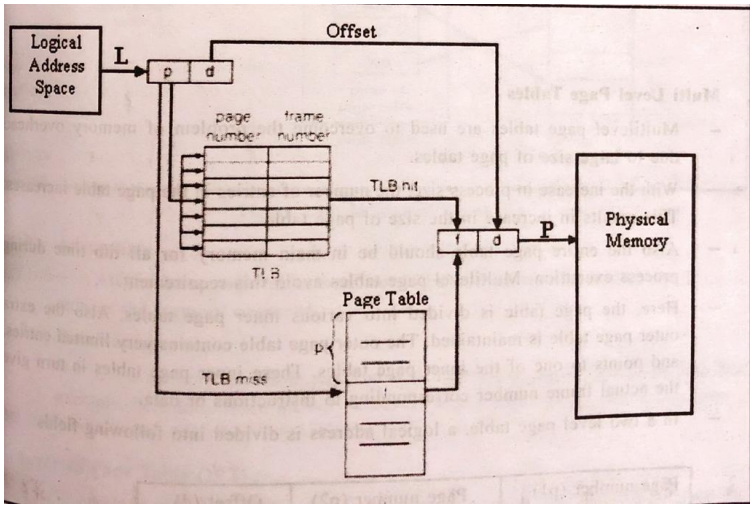
TLB is used to overcome the problem of slower access with basic paging.

TLB is a page-table cache, which is implemented in a fast associative memory.

All the table entries can be searched simultaneously in this memory. This property makes the associative memory much faster than conventional RAM.

But, because of higher cost of associative memory. storage capacity of this memory is limited. So, only a subset of page table is kept in this memory.

The following figure below explains the working of TLB.



Each entry of TLB contains a page number of a page and a frame number where the page is stored in memory.

The TLB works with page table in following manner:

Whenever a logical address is generated, the page number (p) of logical address is searched in TLB.

If a match is found for a page number (p), it is termed as T LB hit. In these cases, the corresponding frame number (f) is fetched from TLB entry and used to get physical address.

If a match is not found, it is termed as TLB miss. In this case, page table is used to get the required frame number. Also, this page table entry moved to TLB, so that, further reference to this page can be satisfied using TLB directly.

If the TLB is full while moving page table entry to TLB, some of the existing entries in TLB are removed based on some page replacement algorithm.

2. Multi-Level Page Tables

Multilevel page tables are used to overcome the problem of memory overhead due to large size of page tables.

With the increase in process size, the number of entries in the page table in the page table increases. This results in increase in the size of page table.

Also, the entire page table should be in main memory for all the time during process execution. Multilevel page tables avoid this requirement.

Here, the page table is divided into various inner page tables. Also, the extra outer page table is maintained. The outer page table contains very limited entries, and points to one of the inner page tables. These inner page tables in turn give the actual frame number corresponding to instructions or data.

In a two-level page table, a logical address is divided into following fields:

Page number (PI)	Page number (p2)	Offset (d)

Here, page number (P1) is used for indexing into the outer page table which points to inner page tables. Page number (P2) is used for indexing into the inner table which gives the frame number containing the intended page.

The following figure depicts this scheme.

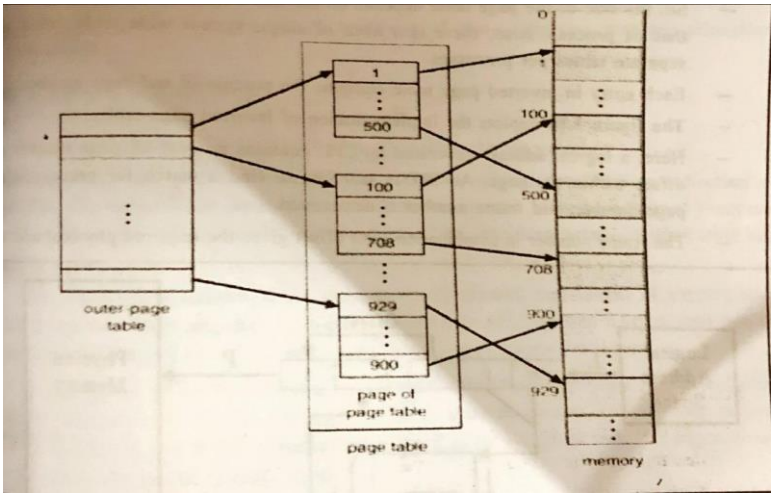


Figure: Multi-level Paging

Advantage:

All the inner page tables need not to be in memory simultaneously. Thus, it reduces the memory overhead.

Disadvantage:

Extra memory access is required with increase in each level of paging. For example, here total three memory accesses are required to get instruction/data.

3. Inverted Page Table (IPT)

Inverted Page Table (IPT) is also used to overcome the problem of memory overhead due to large size of page tables.

Usually, each process has a page table associated with it. Size of this page table depends upon the size of the process. With increase in size of a process, page table size also increases, and it becomes difficult to store in memory.

Inverted page table solves this problem. Here, there is one entry per frame of physical memory in table, rather than one entry per page of logical address space.

So, the size of the page table depends on the size of physical memory rather than that of process. Also, there is a need of single system wide table rather than separate tables per processes.

Each entry in inverted page table contains the process-id and page number.

The figure below depicts the implementation of inverted page table.

Here, a logical address generated by CPU contains process-id, page number and offset within the page. An IPT is searched to find a match for process-id and page number, and frame number is determined.

The frame number in combination with offset gives the required physical address.

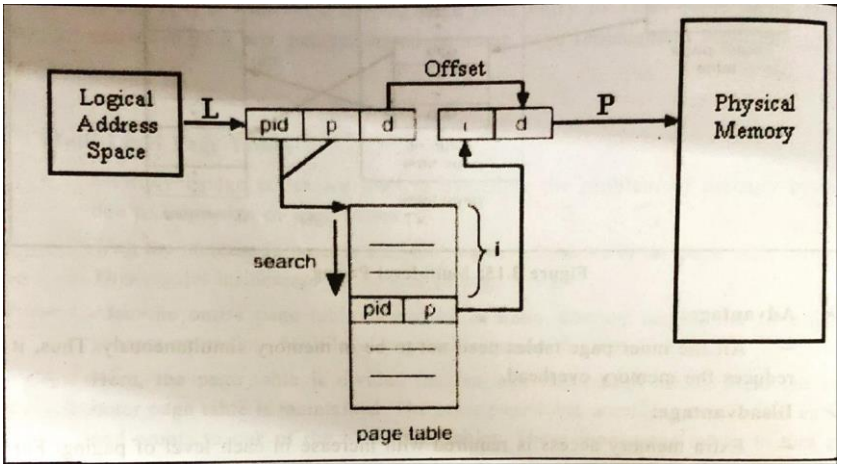


Figure: Inverted Page Table

Advantages:

Only one page table for all processes is required.

Size of page table is constant, and it depends on the size of physical memory.

Disadvantage:

The search time to match an entry in the IPT is large. So, logical-to-physical address translation becomes much slower.

4. Hashed Page Table

Hashed page table handles the address space larger than 32-bits. The virtual page number is used as hashed value. Linked list is used in the hash table.

Each entry contains the linked list of elements that hash to the same location.

Each element in the hash table contains following fields.

1. Virtual page number
2. Mapped page frame value
3. Pointer to the next element in the linked list.

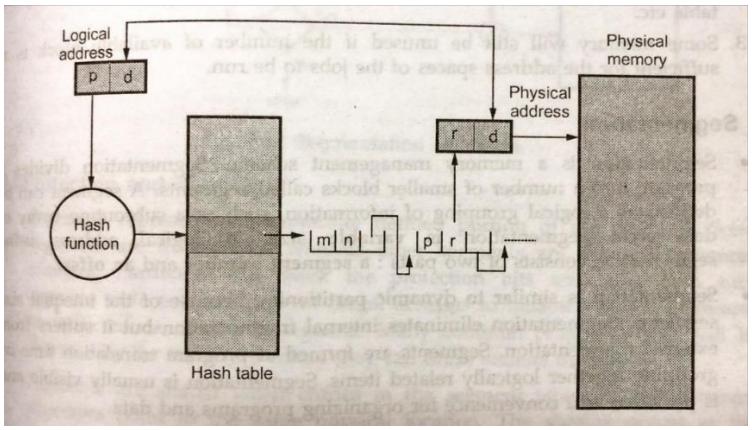


Figure shows hashed page table.

Working

1. Virtual page number is taken from virtual address.

2. Virtual page number is hashed into the hash table.

3. Virtual page number is compared with the first element of linked list.

4. Both the value is matched, that value is (i.e., page frame) used for calculating physical address.

5. If both the values did not match, the entire linked list is searched for a matching.

• Clustered page table is same as hashed page table but only difference is that each entry in the hash table refers to several pages.

4.3.2 Non-contiguous Memory Allocation: Segmentation

This technique implements user's view of logical address space.

From user's point of view, logical address space of any process is a collection of and stack. Code can be comprising of main function, other user defined functions function. Data can be local variables, global variables, arrays. symbol table and data structures.

Here. the logical address space of a process is divided into blocks of varying size, **segments**. Each segment contains a logical unit of a process. Logical unit can be function, other functions or procedures, stack, array, symbol table etc.

Each segment can be considered as a completely independent address space. It consists of linear sequence of addresses starting from '0' to some maximum limit. All segments are of varying lengths and length depends upon the size of a logical unit. All segments are unique numbers to identify them.

Whenever a process is to be executed, its segments are moved from secondary storage i.e. disk, to the main memory. Each segment is allocated a chunk of free memory of the

Size equal to that segment. (This is same as that of dynamic partitioning; but here space is per segments instead of entire process.)

Operating system maintains a table, called **segment table**, for each process. The segment contains information about each segment of a process. This information includes size segment and the **location** in memory where the segment has been loaded.

Here, logical address is divided into **two** parts:

i) **Segment number**, which identifies a segment; and,

ii) **An offset**, which gives the actual location within a segment.

The figure below gives idea about this method. Here, the logical address space of a process has been divided into four segments. A physical memory is shown with these four segments

A segment table is described which gives information such as where each segment is loaded in physical memory and what is the size (length) of each segment.

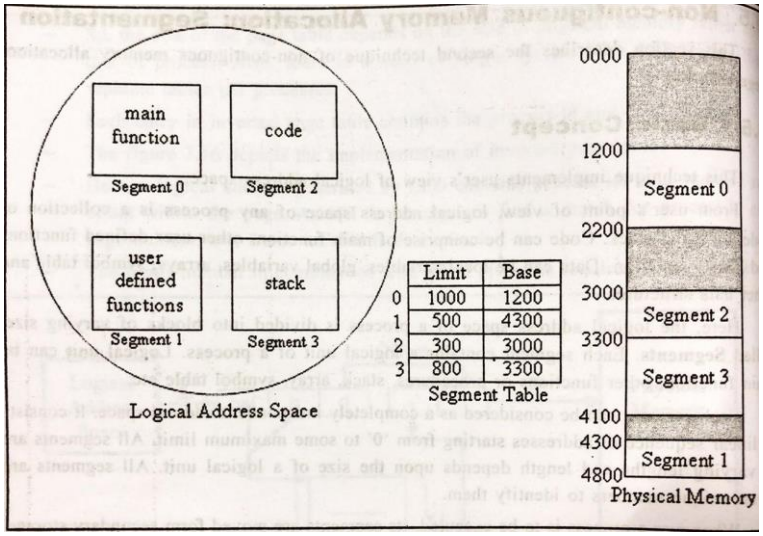


Figure: Basic Concept of Segmentation

Implementation

The figure below depicts how the segmentation technique is implemented. It describes how logical addresses are converted to physical addresses.

A table, called segment table, is used to implement segmentation. Each entry of the segment table has a segment **base** and segment **limit**. The segment base contains the starting physical address where the segment resides in memory. The segment limit specifies the length of the segment.

During the process execution, a CPU generates a logical address (L) to access instructions or data from a particular location. This logical address is divided into two parts: First, a segment number (s), and Second, an offset (d) within that segment.

Logical Address (L):

Segment number (s)	Offset (d)

The segment number is used as an index into the segment table. The offset (d) Of the logical address must be between 0 and the segment limit. If it is not, illegal address error will be generated. Else, if it is within limit, it is added to the segment base to produce the physical address.

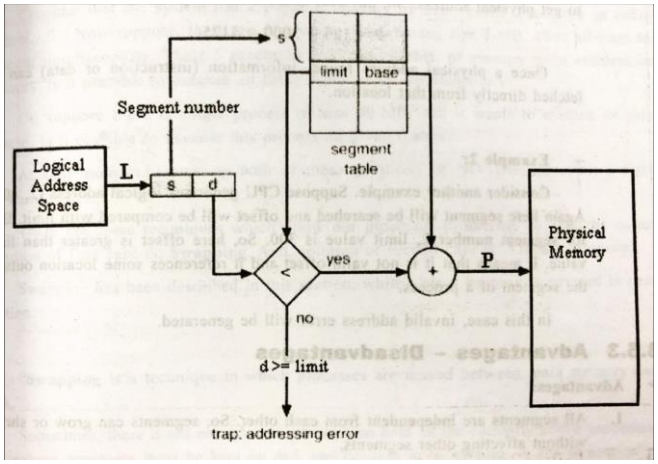


Figure: Address Translation in Segmentation

Advantages:

1. All segments are independent from each other. So, segments can grow or shrink without affecting other segments.

2. If the procedure (or function) in segment 'n' is modified and recompiled, no other segments need to be changed or recompiled.

3. Sharing of procedures and data among various processes is simple.

4. Different segments of a single process can be given different kind of protection. One can be read-only, while other can be writable and so on.

5. There is no Internal Fragmentation. Segments are allocated exactly as much memory as required.

Disadvantages:

1. It is still expensive (or difficult) to allocate contiguous free memory to segments. Some algorithms, such as first fit, is required for allocation of a memory.

2. External Fragmentation is possible, which requires memory de-fragmentation or compaction.

Difference between Contiguous and Non-contiguous Memory Allocation:

Sr No.	Contiguous Memory Allocation	Non-Contiguous Memory Allocation
1.	Contiguous memory allocation allocates consecutive blocks of memory to a file/process.	Non-Contiguous memory allocation allocates separate blocks of memory to a file/process.
2.	Faster in Execution.	Slower in Execution.
3.	It is easier for the OS to control.	It is difficult for the OS to control.
4.	Overhead is minimum as not much address translations are there while executing a process.	More Overheads are there as there are more address translations.
5.	Both Internal fragmentation and external fragmentation occurs in Contiguous memory allocation method.	External fragmentation occurs in Non-Contiguous memory allocation method.
6.	It includes single partition allocation and multi-partition allocation.	It includes paging and segmentation.
7.	Wastage of memory is there.	No memory wastage is there.
8.	In contiguous memory allocation, swapped-in processes are arranged in the originally allocated space.	In non-contiguous memory allocation, swapped-in processes can be arranged in any place in the memory.

Difference between Paging and Segmentation:

Sr No	Paging	Segmentation
1	Program is divided into variable size segments.	Program is divided into fixed size pages.
2	User (or compiler) is responsible for dividing the program into segments.	Division into pages is performed by the operating system.
3	Segmentation is slower than paging.	Paging is faster than segmentation.
4	Segmentation is visible to the user.	Paging is invisible to the user.
5	Segmentation eliminates internal fragmentation.	Paging suffers from internal fragmentation.
6	Segmentation suffers from external fragmentation.	There is no external fragmentation.
7	Processor uses page number, offset to calculate absolute address.	Processor uses segment number, offset to calculate absolute address.
8	OS maintain a list of free holes in main memory.	OS must maintain a free frame list.