Chapter 4

- Basic Concepts of Memory Management
- Swapping
- Contiguous Memory Allocation
- Paging
- Structure of Page Table
- Segmentation
- Basic Concepts of Virtual Memory
- Demand Paging
- Copy-on Write
- Page Replacement Algorithms
- Thrashing.

➢ Basic Concepts of Memory Management

The term Memory can be defined as a collection of data in a specific format. It is used to store instructions and process data. The memory comprises a large array or group of words or bytes, each with its own location. The primary motive of a computer system is to execute programs. These programs, along with the information they access, should be in the main memory during execution. The CPU fetches instructions from memory according to the value of the program counter.

To achieve a degree of multiprogramming and proper utilization of memory, memory management is important. Many memory management methods exist, reflecting various approaches, and the effectiveness of each algorithm depends on the situation.

Main memory is the place where programs and information are kept when the processor is effectively utilizing them. Main memory is associated with the processor, so moving instructions and information into and out of the processor is extremely fast. Main memory is also known as RAM(Random Access Memory). This memory is a volatile memory. RAM lost its data when a power interruption occurs.

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.

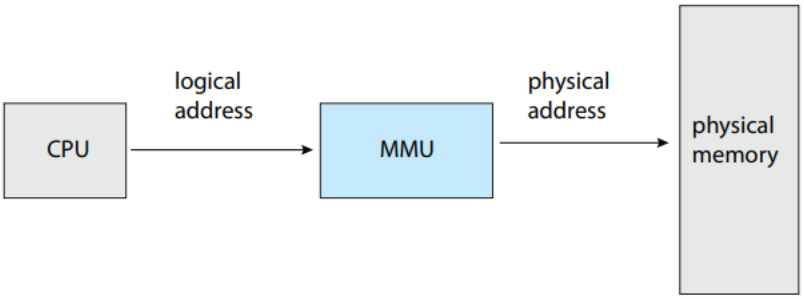


Figure 9.4 Memory management unit (MMU).

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

1. **Static loading**:- loading the entire program into a fixed address. It requires more memory space.
2. **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 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.

1. **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.
2. **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

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 as 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, roll in, 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.

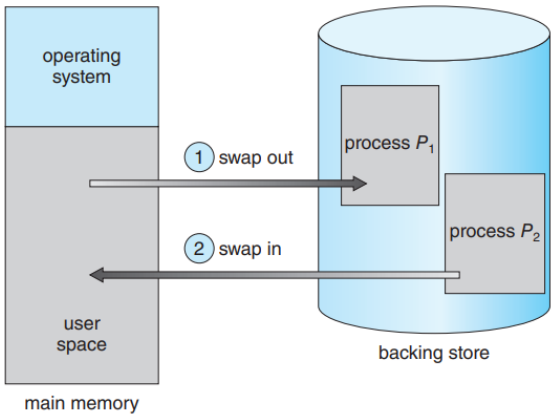


Figure 9.19 Standard swapping of two processes using a disk as a backing store.

1. **Standard Swapping** Standard swapping involves moving entire processes between main memory and a backing store. The backing store is commonly fast secondary storage. It must be large enough to accommodate whatever parts of processes need to be stored and retrieved, and it must provide direct access to these memory images. When a process or part is swapped to the backing store, the data structures associated with the process must be written to the backing store. For a multithreaded process, all per-thread data structures must be swapped as well. The operating system must also maintain metadata for processes that have been swapped out, so they can be restored when they are swapped back in to memory.

2. **Swapping with paging** Most systems, including Linux and Windows, now use a variation of swap□ping in which pages of a process—rather than an entire process—can be swapped. This strategy still allows physical memory to be oversubscribed, but does not incur the cost of swapping entire processes, as presumably only a small number of pages will be involved in swapping. In fact, the term swapping now generally refers to standard swapping, and paging refers to swapping with paging.

➢ Contiguous Memory Allocation

The main memory should oblige 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.

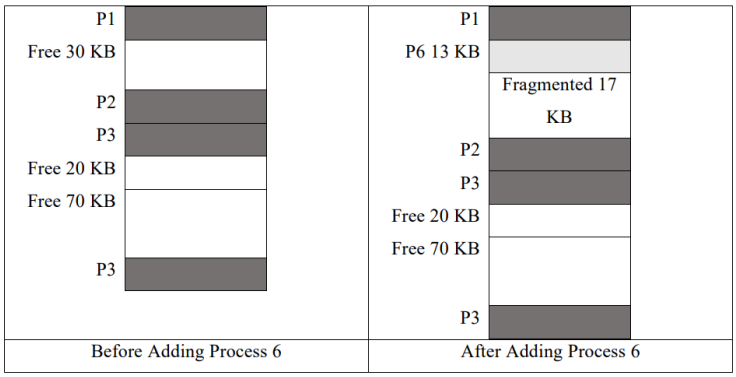
Memory allocation:

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.

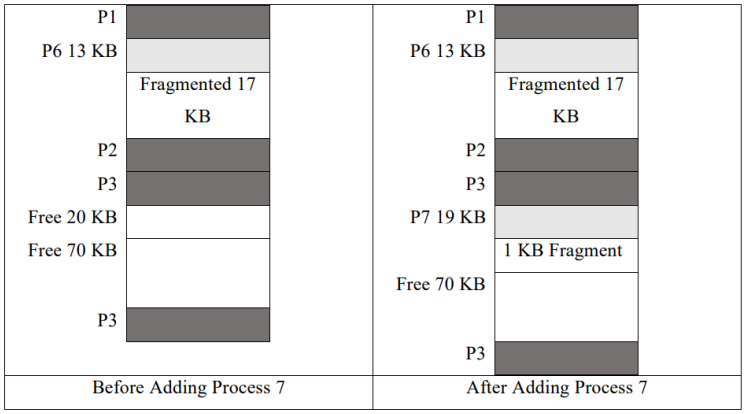
Multiple partition allocation: In this method, a process is selected from the input queue and loaded into the free partition. When the process terminates, the partition becomes available for other processes.

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:

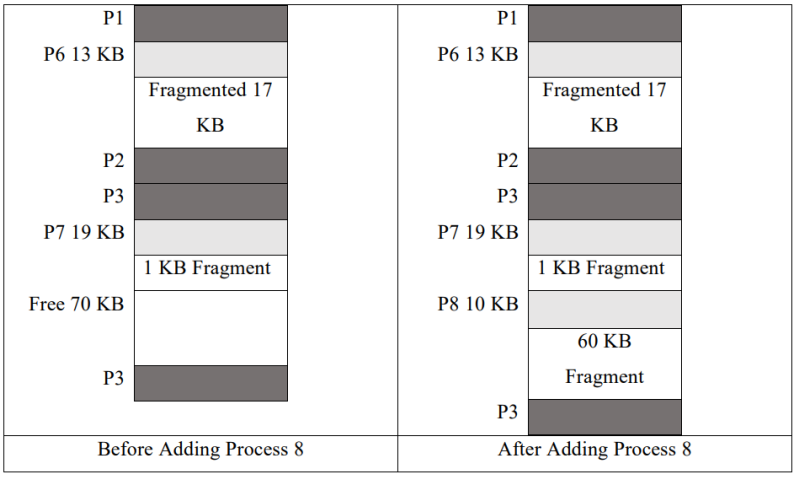
1. First fit. Allocate the first hole that is big enough. Searching can start either at the beginning of the set of holes or at the location where the previous first-fit search ended. We can stop searching as soon as we find a free hole that is large enough.



2. Best fit . Allocate the smallest hole that is big enough. We must search the entire list, unless the list is ordered by size. This strategy produces the smallest leftover hole.



3. Worst fit. Allocate the largest hole. Again, we must search 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.



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:

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 leftover and creates an internal fragmentation problem.

Example: Suppose there is a fixed partitioning is used for memory allocation and the different size of block 3MB, 6MB, and 7MB space in memory. Now a new process p4 of size 2MB comes and demand for the block of memory. It gets a memory block of 3MB but 1MB

block memory is a waste, and it can not be allocated to other processes too. This is called internal fragmentation.

External fragmentation:
In external fragmentation, we have a free memory block, but we can not assign it to process because blocks are not contiguous.

Example: Suppose (consider above example) three process p1, p2, p3 comes with size 2MB, 4MB, and 7MB respectively. Now they get memory blocks of size 3MB, 6MB, and 7MB allocated respectively. After allocating process p1 process and p2 process left 1MB and 2MB. Suppose a new process p4 comes and demands a 3MB block of memory, which is available, but we can not 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 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 permit a process to be allocated physical memory wherever the latter is available.

➢ Paging

Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory. The process of retrieving processes in the form of pages from the secondary storage into the main memory is known as paging. The basic purpose of paging is to separate each procedure into pages. Additionally, frames will be used to split the main memory. This scheme permits the physical address space of a process to be non – contiguous. some important terminologies of paging are

- 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 memory unit.
- Physical Address Space (represented in words or bytes): The set of all physical addresses corresponding to the logical addresses

Features of paging:

1. Mapping logical address to physical address.
2. Page size is equal to frame size.
3. Number of entries in a page table is equal to number of pages in logical address space.
4. The page table entry contains the frame number.
5. All the page table of the processes are placed in main memory.

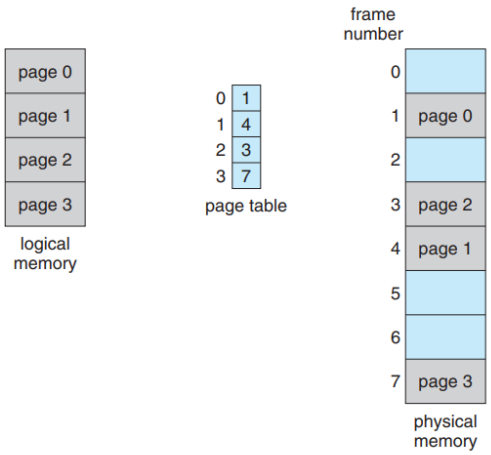


Figure 9.9 Paging model of logical and physical memory.

1. **Page number(p):** Number of bits required to represent the pages in Logical Address Space or Page number
2. **Page offset(d):** Number of bits required to represent particular word in a page or page size of Logical Address Space or word number of a page or page offset.
3. **Frame number(f):** Number of bits required to represent the frame of Physical Address Space or Frame number.
4. **Frame offset(d):** Number of bits required to represent particular word in a frame or frame size of Physical Address Space or word number of a frame or frame offset.

➢ **Translation Lookaside Buffer (TLB)**

Size of Page table can be very big and therefore it wastes main memory. CPU will take more time to read a single word from the main memory.

The page table size can be decreased by increasing the page size but it will cause internal fragmentation and there will also be page wastage. Other way is to use multilevel paging but that increases the effective access time therefore this is not a practical approach.

CPU can use a register having the page table stored inside it so that the access time to access page table can become quite less but the register are not cheaper and they are very small in compare to the page table size therefore, this is also not a practical approach. To overcome these many drawbacks in paging, we have to look for a memory that is cheaper than the register and faster than the main memory so that the time taken by the CPU to access page table again and again can be reduced and it can only focus to access the actual word.

A Translation look aside buffer can be defined as a memory cache which can be used to reduce the time taken to access the page table again and again. It is a memory cache which is closer to the CPU and the time taken by CPU to access TLB is lesser then that taken to access main memory. In other words, we can say that TLB is faster and smaller than the main memory but cheaper and bigger than the register.

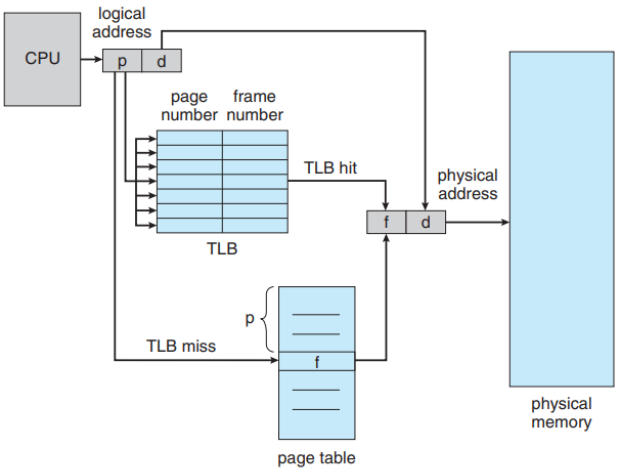


Figure 9.12 Paging hardware with TLB.

In translation look aside buffers, there are tags and keys with the help of which, the mapping is done. TLB hit is a condition where the desired entry is found in translation look aside buffer. If this happens then the CPU simply access the actual location in the main memory.

However, if the entry is not found in TLB (TLB miss) then CPU has to access page table in the main memory and then access the actual frame in the main memory. Therefore, in the case of TLB hit, the effective access time will be lesser as compare to the case of TLB miss.

- **Structure of Page Table**

  1. Inverted Page Table

  The **Inverted Page Table** structure that consists of a one-page table entry for every frame of the main memory. So the number of page table entries in the Inverted Page Table reduces to the number of frames in physical memory and a single page table is used to represent the paging information of all the processes. Through the inverted page table, the overhead of storing an individual page table for every process gets eliminated and only a fixed portion of memory is required to store the paging information of all the processes together. This technique is called inverted paging as the indexing is done with respect to the frame number instead of the logical page number.

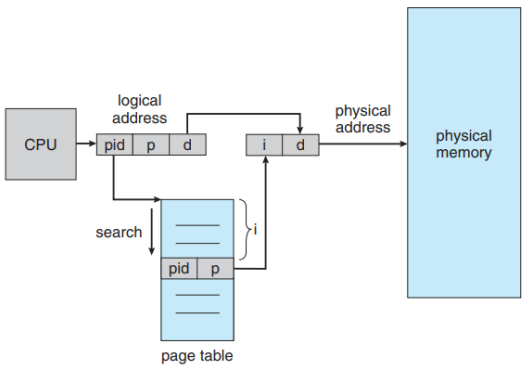


Figure 9.18 Inverted page table.

Advantages and Disadvantages:

1. **Reduced memory space –** Inverted Page tables typically reduce the amount of memory required to store the page tables to a size bound of physical memory.
2. **Longer lookup time –** Inverted Page tables are sorted in order of frame number but the memory look-up takes place with respect to the virtual address, so, it usually takes a longer time to find the appropriate entry but often these page tables are implemented using hash data structures for a faster lookup.
3. **Difficult shared memory implementation –** As the Inverted Page Table stores a single entry for each frame, it becomes difficult to implement the shared memory in the page tables.
4. **Optimal and less complex –** it is better than simple paging process and have less complexity.

2. Hierarchical Page Table

Multilevel Paging is a paging scheme that consists of two or more levels of page tables in a hierarchical manner. It is also known as hierarchical paging. The entries of the level 1 page table are pointers to a level 2 page table and entries of the level 2 page tables are pointers to a level 3 page table and so on. The entries of the last level page table store actual frame information. Level 1 contains a single-page table and the address of that table is stored in PTBR (Page Table Base Register).

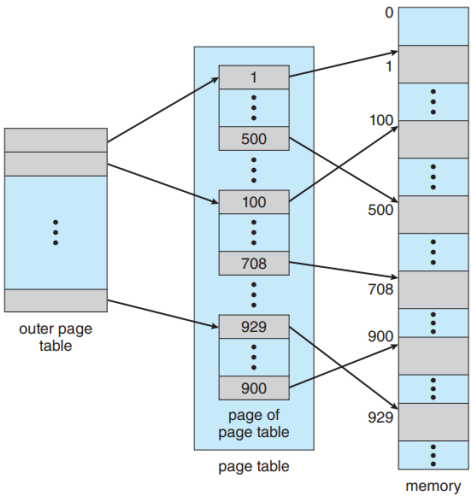


Fig.: Overview of two -level Page table

Disadvantage:

Extra memory references to access address translation tables can slow programs down by a factor of two or more. Use translation look aside buffer (TLB) to speed up address translation by storing page table entries.

3. Hashed Page Table

In hashed page tables, the virtual page number in the virtual address is hashed into the hash table. They are used to handle address spaces higher than 32 bits. Each entry in the hash table has a linked list of elements hashing to the same location (to avoid collisions – as we can get the same value of a hash function for different page numbers). The hash value is the virtual page number. The Virtual Page Number is all the bits that are not a part of the page offset.

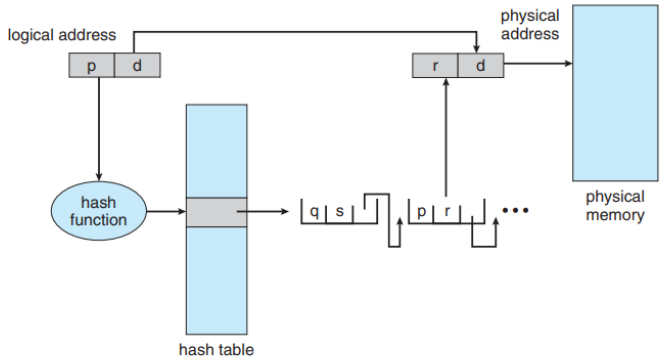


Figure 9.17 Hashed page table.

➢ Segmentation

Segmentation divides processes into smaller subparts known as **modules**. The divided segments need not be placed in contiguous memory. Since there is no contiguous memory allocation, internal fragmentation does not take place. The length of the segments of the program and memory is decided by the purpose of the segment in the user program.

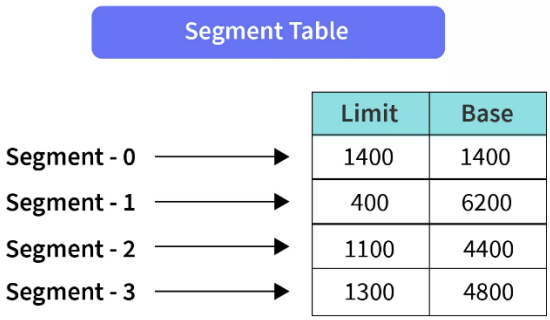
The actual view of physical memory is separated from the user's view of physical memory. Segmentation in OS helps in overcoming the problem by dividing the user's program into segments according to the specific need.

Segment Table and Its Uses: Segment Table is used to store the information of all segments of the process. As we know, the CPU generates a logical address and for its conversion to the physical address, a segment table is used. The mapping of a two-dimensional Logical address to a one-dimensional Physical address is done using the segment table.

The Segment Table has two components:

1. **Segment Base**: The segment base is also known as the base address of a segment. The segment base contains the starting physical address of the segments residing in the memory.
2. **Segment Limit**: The segment limit is also known as segment offset. The segment contains the specific length of the segment.

The basic overview of the Segment Table is shown below.



- **STBR**: STBR stands for Segment Table Base Register. STBR stores the base address of the segment table.
- **STLR**: STLR stands for Segment Table Length Register. STLR stores the number of segments used by a program.
- **NOTE:** The segment table is itself stored as a separate segment in the main memory. The segment table can sometimes consume large memory if there are several segments.

Types of Segmentation in OS

1. **Virtual Memory Segmentation**: Virtual Memory Segmentation divides the processes into **n** number of segments. All the segments are not divided at a time. Virtual Memory Segmentation may or may not take place at the run time of a program.
2. **Simple Segmentation**: Simple Segmentation also divides the processes into **n** number of segments but the segmentation is done all together at once. Simple segmentation takes place at the run time of a program. Simple segmentation may scatter the segments into the memory such that one segment of the process can be at a different location than the other(in a noncontinuous manner).

Characteristics of Segmentation in OS

- Segmentation partitions the program into variable-sized blocks or segments.
- Partition size depends upon the type and length of modules.
- Segmentation is done considering that the relative data should come in a single segment.
- Segments of the memory may or may not be stored in a continuous manner depending upon the segmentation technique chosen.
- Operating System maintains a segment table for each process.

➢ Basic Concepts of Virtual Memory

Virtual Memory is a storage allocation scheme in which secondary memory can be addressed as though it were part of the main memory. The addresses a program may use to reference memory are distinguished from the addresses the memory system uses to identify physical storage sites, and program-generated addresses are translated automatically to the corresponding machine addresses.

The size of virtual storage is limited by the addressing scheme of the computer system and the amount of secondary memory is available not by the actual number of the main storage locations.

It is a technique that is implemented using both hardware and software. It maps memory addresses used by a program, called virtual addresses, into physical addresses in computer memory.

1. All memory references within a process are logical addresses that are dynamically translated into physical addresses at run time. This means that a process can be swapped in and out of the main memory such that it occupies different places in the main memory at different times during the course of execution.
2. A process may be broken into a number of pieces and these pieces need not be continuously located in the main memory during execution. The combination of dynamic run-time address translation and use of page or segment table permits this.

➢ Demand Paging

- According to the concept of Virtual Memory, in order to execute some process, only a part of the process needs to be present in the main memory which means that only a few pages will only be present in the main memory at any time.
- However, deciding, which pages need to be kept in the main memory and which need to be kept in the secondary memory, is going to be difficult because we cannot say in advance that a process will require a particular page at particular time.

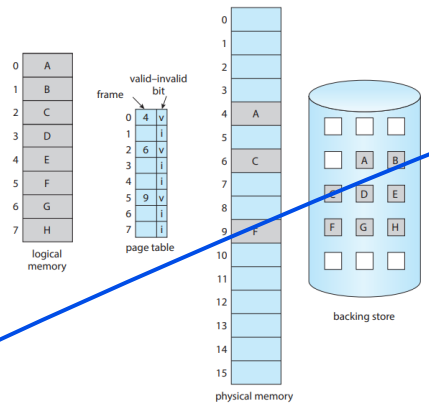


Figure 10.4 Page table when some pages are not in main memory.

- Therefore, to overcome this problem, there is a concept called Demand Paging is introduced. It suggests keeping all pages of the frames in the secondary memory until they are required. In other words, it says that do not load any page in the main memory until

it is required. Whenever any page is referred for the first time in the main memory, then that page will be found in the secondary memory.

➢ Copy-on Write

Copy on Write is a resource management technique. One of its main use is in the implementation of the fork system call in which it shares the virtual memory(pages) of the OS.
In UNIX like OS, fork() system call creates a duplicate process of the parent process which is called as the child process.

The idea behind a copy-on-write is that when a parent process creates a child process then both of these processes initially will share the same pages in memory and these shared pages will be marked as copy-on-write which means that if any of these processes will try to modify the shared pages then only a copy of these pages will be created and the modifications will be done on the copy of pages by that process and thus not affecting the other process.

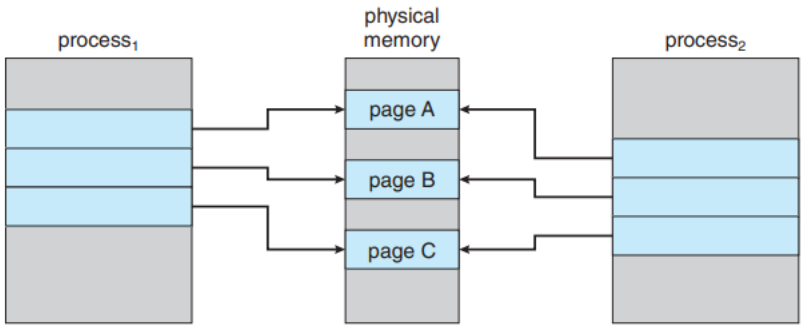


Figure 10.7 Before process 1 modifies page C.

➢ Page Replacement Algorithms

A page fault happens when a running program accesses a memory page that is mapped into the virtual address space but not loaded in physical memory. Since actual physical memory is much smaller than virtual memory, page faults happen. In case of a page fault, Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults.

1. **First In First Out (FIFO):** This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

   Example : Consider page reference string 1, 3, 0, 3, 5, 6, 3 with 3 pages frames. Find the number of page faults.

1	3	0	3	5	6	3
1	1	1	1	5	5	5
	3	3	3	3	6	6
		0	0	0	0	3
Hit	Hit	Hit	Miss	Hit	Hit	Hit

Total Page fault: 6

2. **Least Recently Used:** In this algorithm, page will be replaced which is least recently used.

   Example-: Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4 page frames. Find number of page faults.

7	0	1	2	0	3	0	4	2	3	0	3	2	3
7	7	7	7	7	3	3	3	3	3	3	3	3	3
	0	0	0	0	0	0	0	0	0	0	0	0	0
		1	1	1	1	1	4	4	4	4	4	4	4
			2	2	2	2	2	2	2	2	2	2	2
Hit	Hit	Hit	Hit	Miss	Hit	Miss	Hit	Miss	Miss	Miss	Miss	Miss	Miss

Total Page Fault: 6

3. **Optimal Page replacement:** In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

   Example: Consider the page references 6, 1, 1, 2, 0, 3, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1, 2, 0 with 4 page frame. Find number of page fault.

6	1	1	2	0	3	4	6	0	2	1	2	1	2	0	3	2	1	2
6	6	6	6	6	6	6	6	6	6	1	1	1	1	1	1	1	1	1
	1	1	1	1	3	4	4	4	4	4	4	4	4	4	3	3	3	3
			2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
				0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
H	M	M	H	H	H	H	M	M	M	H	M	M	M	M	H	M	M	M

Total Page Fault: 07

➢ Thrashing

Thrashing is a condition or a situation when the system is spending a major portion of its time servicing the page faults, but the actual processing done is very negligible.

Causes of thrashing:

1. High degree of multiprogramming.
2. Lack of frames.

3. Page replacement policy.

Thrashing's Causes

Thrashing has an impact on the operating system's execution performance. Thrashing also causes serious performance issues with the operating system. When the CPU's usage is low, the process scheduling mechanism tries to load multiple processes into memory at the same time, increasing the degree of Multi programming.

In this case, the number of processes in the memory exceeds the number of frames available in the memory. Each process is given a set number of frames to work with.

If a high-priority process arrives in memory and the frame is not vacant at the moment, the other process occupying the frame will be moved to secondary storage, and the free frame will be allotted to a higher-priority process.

Technique to Handle Thrashing: -
Working set model
This model is based on the concept of the Locality Model.
The basic principle states that if we allocate enough frames to a process to accommodate its current locality, it will only fault whenever it moves to some new locality. But if the allocated frames are lesser than the size of the current locality, the process is bound to thrash.

Page Fault frequency
Thrashing has a high page-fault rate.
We want to control the page-fault rate.
When it is too high, the process needs more frames. Conversely, if the page-fault rate is too low, then the process may have too many frames.
We establish upper and lower bounds on the desired page fault rate.
If pf-rate exceeds the upper limit, allocate the process another frame, if pf-rate fails falls below the lower limit, remove a frame from the process.
By controlling pf-rate, thrashing can be prevented.