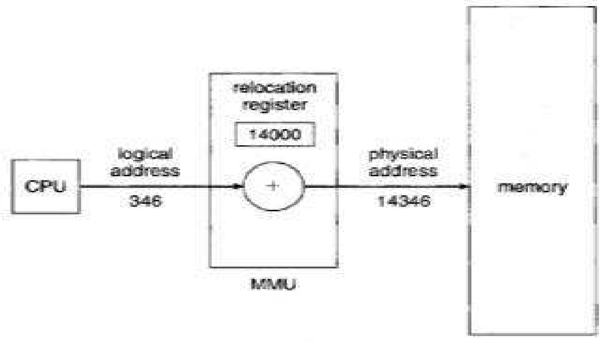
Unit-4

Memory Management Strategies

The memory management algorithms vary from a primitive bare-machine approach to paging and segmentation strategies. Each approach has its own advantages and disadvantages. Selection of a memory-management method for a specific system depends on many factors, especially on the hardware design of the system.

Logical versus Physical Address Space

- An address generated by the CPU is commonly referred to as a **Logical Address.**
- An address seen by the memory unit is commonly referred to as a **Physical Address.**
- The set of all logical addresses generated by a program is a logical address space.
- The set of all physical addresses corresponding to these logical addresses is a physical address space.
- The run-time mapping from virtual to physical addresses is done by a hardware device called Memory Management Unit (MMU).



Fragmentation

External Fragmentation: External fragmentation exists when there is enough total memory space to satisfy a request but the available spaces are not contiguous; storage is fragmented into a large number of small holes. This fragmentation problem can be severe. In the worst case, we could have a block of free (or wasted) memory between every two processes. If all these small pieces of memory were in one big free block instead, we might be able to run several more processes.

Internal Fragmentation: The memory allocated to a process may be slightly larger than the requested memory. The difference between these two numbers is internal memory that is internal to a partition.

<u>Compaction:</u> One solution to the problem of external fragmentation is Compaction. The goal is to shuffle the memory contents so as to place all free memory together in one large block. Compaction is not always possible. The simplest compaction algorithm is to move all processes toward one end of memory; all holes move in the other direction, producing one large hole of available memory but this scheme can be expensive. Another possible solution to the external-fragmentation problem is to permit the logical address space of the processes to be noncontiguous.

Contiguous memory allocation

- ❖ In contiguous memory allocation, each process is contained in a single contiguous section of memory.
- ❖ Accessing memory in contiguous memory allocation is faster.
- ❖ Managing contiguous memory allocation is easier.
- ❖ External fragmentation may occur in contiguous memory allocation.
- ❖ Contiguous memory allocation can be classified in following two categories:
    1. MFT (Multiprogramming with Fixed number of tasks).
    2. MVT (Multiprogramming with variable number of tasks).

MFT (Multiprogramming with Fixed number of tasks)

In this method memory is divided into fixed sized partitions. Each partition may contain exactly one process. Thus, the degree of multiprogramming is bound by the number of partitions. In this multiple partition method when a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process. MFT suffers from following problems—

- ✓ Internal fragmentation may occur in this method.
- ✓ External fragmentation may occur in this method.
- ✓ Deciding most appropriate value of Number of partitions and size of partition is difficult.

MVT (Multiprogramming with variable number of tasks)

In the scheme, 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 one large block of available memory, a hole.

As processes enter the system, they are put into an input queue. The operating system takes into account the memory requirements of each process and the amount of available memory space in determining which processes are allocated memory. When a process is allocated space, it is loaded into memory, and it can then compete for CPU time. When a process terminates, it releases its memory which the operating system may then fill with another process from the input queue.

- ✓ This method solves the problem of internal fragmentation.
- ✓ External fragmentation may occur in this method.

Algorithms used in contiguous memory allocation techniques

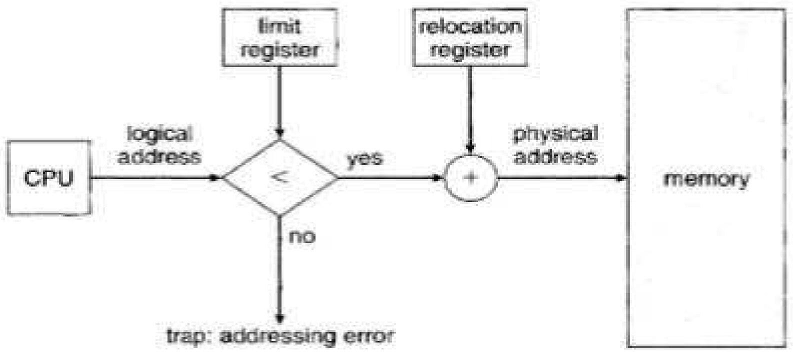
First fit: Allocate the first hole that is big enough. Searching can start either at the beginning of the set of holes or at the location where the previous first-fit search ended. We can stop searching as soon as we find a free hole that is large enough.

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.

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.

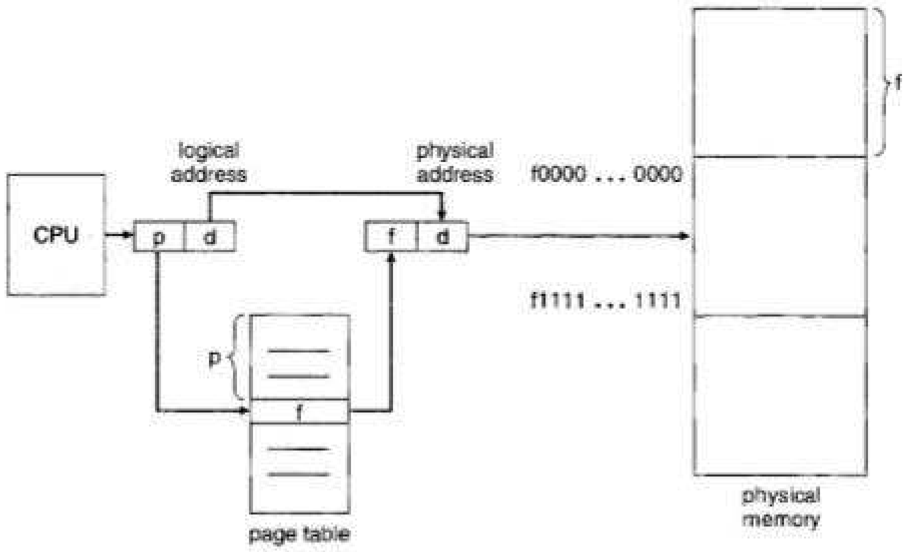
Memory Protection in contiguous memory allocation

- ❖ To make sure that each process has a separate memory space, we need the ability to determine the range of legal addresses that the process may access and to ensure that the process can access only these legal addresses. We can provide this protection by using two registers, usually a base and a limit or relocation.
- ❖ The base register holds the smallest legal physical memory address. The limit register specifies the size of the range. For example, if the base register holds 300 and the limit register is 100, then the program can legally access all addresses from 300 through 400.
- ❖ Protection of memory space is accomplished by having the CPU hardware compare every address generated in user mode with the registers. Any attempt by a program executing in user mode to access operating-system memory or other users' memory results in a trap to the operating system.



Paging

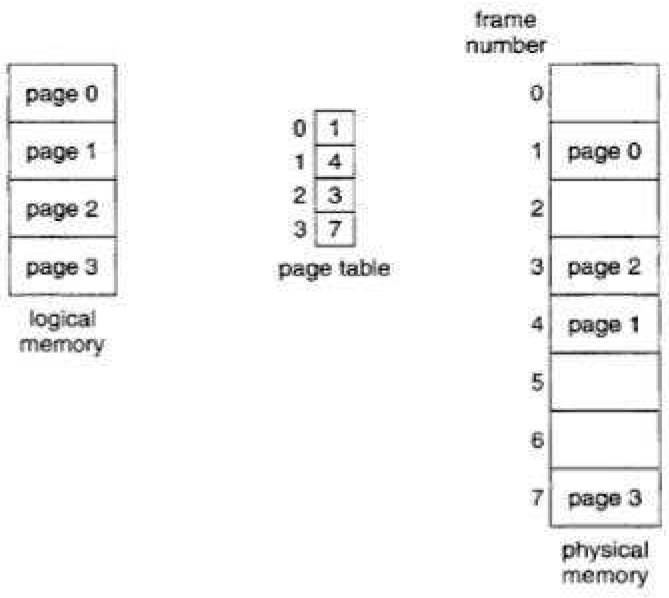
- ❖ Paging is a memory-management scheme that permits the physical address space a process to be non-contiguous.
- ❖ Paging avoids external fragmentation but not internal fragmentation.



Paging hardware.

- ❖ Paging involves breaking physical memory into fixed-sized blocks called frames and breaking logical memory into blocks of the same size called pages.
- ❖ When a process is to be executed, its pages are loaded into any available memory frames from their source
- ❖ Every address generated the CPU (Logical address) is divided into two parts: a page number p and a page offset d. The page number is used as an index into a page table. The page table contains the base address of each page in physical memory. This base address is combined with the page offset to define the physical memory address that is sent to the memory unit.
- ❖ The size of a page is typically a power of 2, varying between 512 bytes and 16 MB per page, depending on the computer architecture. The selection of a power of 2 as a page size makes the translation of a logical address into a page number and page offset particularly easy.

Example:



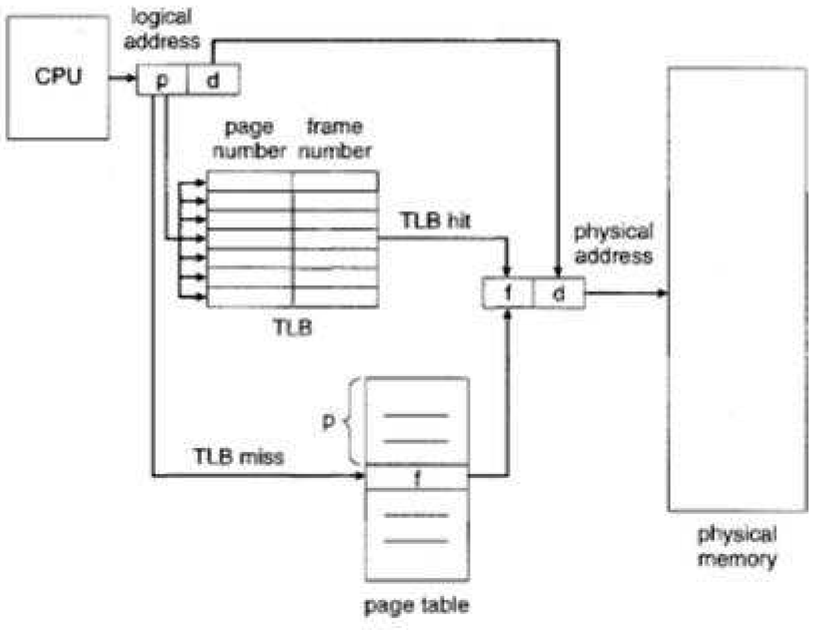
Paging model of logical and physical memory.

Page Table

- ✓ Page table is a data structure maintained by OS.
- ✓ In paging OS creates page table for every process separately.
- ✓ Page table contains frame number, corresponding to Page number.
- ✓ Page table resides in physical memory; it wastes large space in physical memory because generally its size is large.

Paging with TLB

- To increase the speed of memory access, TLB is used with page table.
- TLB is a special, small, fast, lookup hardware cache, called a translation look aside buffer.
- The TLB is used with page tables in the following way.
  - ✓ The TLB contains only a few of the page-table entries. When a logical address is generated by the CPU, its page number is presented to the TLB. If the page number is found (known as a TLB Hit), its frame number is immediately available and is used to access memory.
  - ✓ If the page number is not in the TLB (known as a TLB Miss), a memory reference to the page table must be made. When the frame number is obtained, we can use it to access memory.



Paging hardware with TLB.

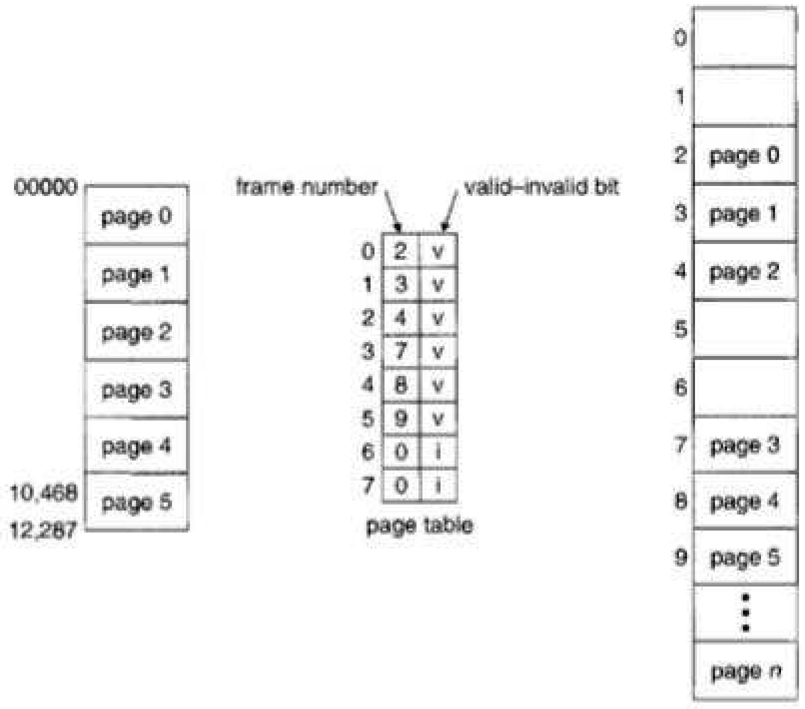
- **Hit Ratio p = $N_{Hit}/(N_{Hit} + N_{Miss})$**
- **Effective Memory Access time = $(N_{Hit} *(T_{TLB} + T_{PM}) + N_{Miss} *(T_{TLB} + 2*T_{PM}))/(N_{Hit} + N_{Miss})$**

Where:-

N_{Miss} → Total number of Miss in TLB
N_{Hit} → Number of Hit in TLB
T_{TLB} → access time of TLB
T_{PM} → access time of PM

Protection in Paging

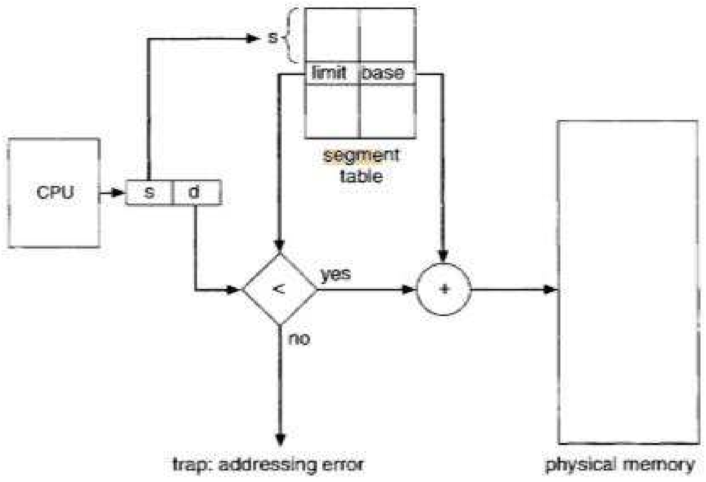
- Memory protection in a paged environment is accomplished by protection bits associated with each frame. Normally, these bits are kept in the page table.
- One bit can define a page to be read-write or read-only. Every reference to memory goes through the page table to find the correct frame number. At the same time that the physical address is being computed, the protection bits can be checked to verify that no writes are being made to a read-only page. An attempt to write to a read-only page causes a hardware trap to the operating system.
- One additional bit is generally attached to each entry in the page table called valid-invalid bit. When this bit is set to "valid," the associated page is in the process's logical address space and is thus a legal (or valid) page. When the bit is set to “invalid," the page is not in the process's logical address space. Illegal addresses are trapped by use of the valid -invalid bit. The operating system sets this bit for each page to allow or disallow access to the page.



Valid (v) or invalid (i) bit in a page table.

Segmentation

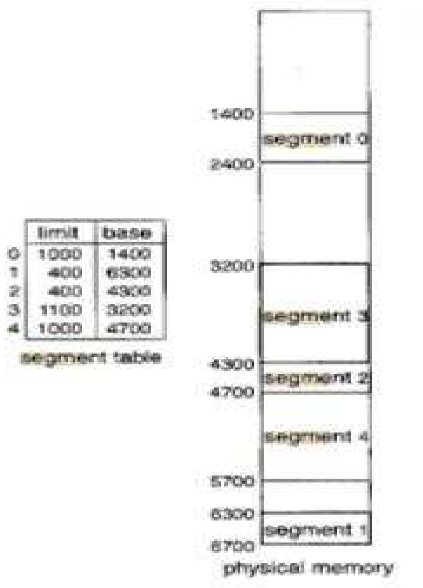
- Segmentation is a memory-management scheme that supports user view of memory.
- In segmentation logical address space is a collection of variable size segments. Each segment has a name and a length.
- Segmentation avoids internal fragmentation and external fragmentation both.
- Segment table takes lesser space in physical memory than page table.



- Each entry in the segment table has a segment base and a segment limit. The segment base contains the starting physical address where the segment resides in memory, and the segment limit specifies the length of the segment.

- In segmentation logical address consists of two parts: a segment number, s, and an offset into that segment, d. the segment number are used as an index to the segment table. The offset d of the logical address must be between 0 and the segment limit. If it is not, we trap to the operating system (logical addressing attempt beyond end of segment). When an offset is legal, it is added to the segment base to produce the address in physical memory of the desired byte. The segment table is thus essentially an array of base-limit register pairs.

  Example: segment 2 is 400 bytes long and begins at location 4300. Thus, a reference to byte 53 of segment 2 is mapped onto location 4300 +53= 4353. A reference to segment 3, byte 852, is mapped to 3200 (the base of segment 3) + 852 = 4052. A reference to byte 1222 of segment 0 would result in a trap to the operating system, as this segment is only 1,000 bytes long.



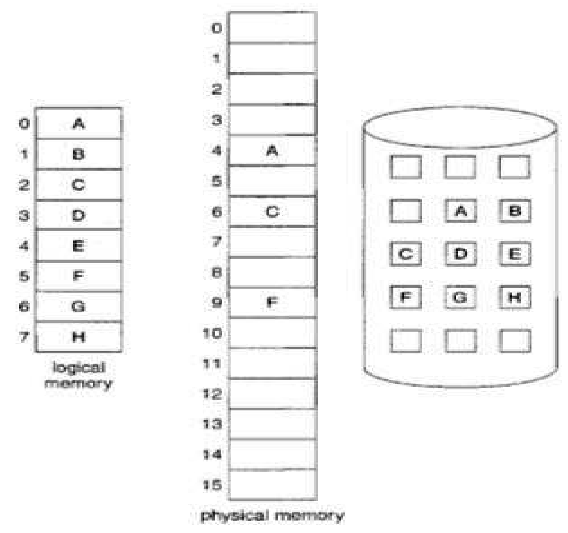
	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

Virtual memory

- Virtual memory is a technique that allows the execution of processes that are not completely in memory. One major advantage of this scheme is that programs can be larger than physical memory.
- Virtual memory also allows processes to share files easily and to implement shared memory.
- Virtual memory is not easy to implement, however, and may substantially decrease performance if it is used carelessly.

Demand Paging

- In demand-paged virtual memory, pages are only loaded when they are demanded during program execution. Pages that are never accessed are thus never loaded into physical memory.
- This technique is commonly used in virtual memory systems.
- In demand paging lazy swapper is used. It never swaps a page into memory unless that page will be needed.
- Objective of Demand paging is to increase degree of multi-programming.



Performance of Demand Paging: Performance of demand paging is generally measured by effective access time. Let p be the probability of a page fault ($0 \leq p \leq 1$). We would expect p to be close to zero-that is, we would expect to have only a few page faults. The effective access time is then-

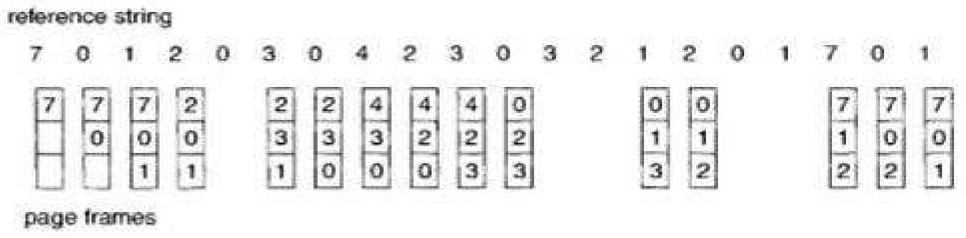
Effective access time= (1 - p) * ma + p * page fault time.

Page Replacement Policies

1. FIFO Page Replacement:

- ❖ The simplest page-replacement algorithm is a first-in, first-out (FIFO) algorithm. A FIFO replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen.
- ❖ FIFO page-replacement algorithm is easy to understand and program.
- ❖ Belady's anomaly may occurs for some sequence like 1,2,3,4,1,2,5,1,2,3,4,5 number of faults for four frames (ten) is greater than the number of faults for three frames (nine).

Example:

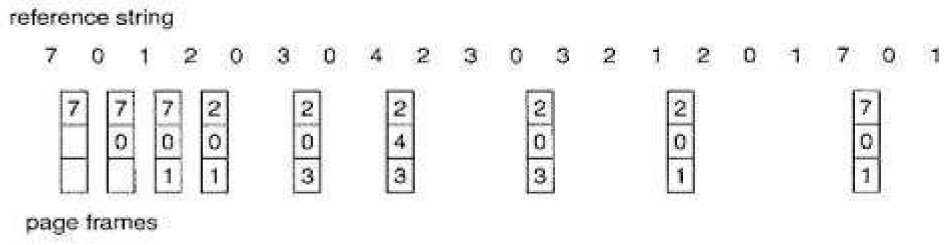


2. Optimal Page Replacement:

- ❖ It has the lowest page-fault rate of all algorithms and will never suffer from Belady's anomaly.
- ❖ It Replace the page that will not be used for the longest period of time.

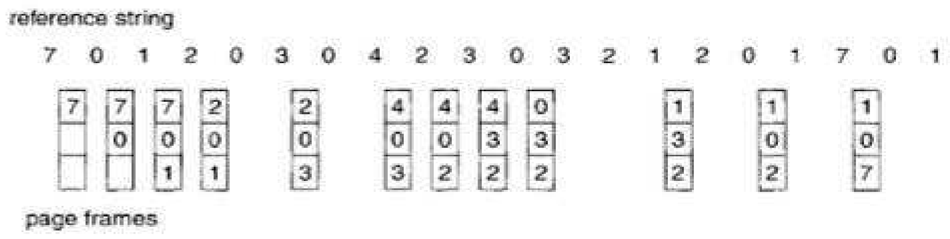
- Use of this page-replacement algorithm guarantees the lowest possible page fault rate for a fixed number of frames.
- The optimal page-replacement algorithm is difficult to implement, because it requires future knowledge of the reference string.

Example:



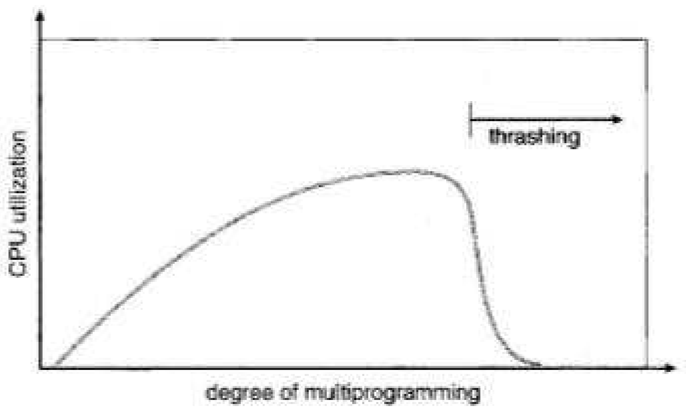
3. **LRU Page Replacement:**
    - In least recently algorithm we can replace the page that has not been used for the longest period of time.
    - The major problem is how to implement LRU replacement. An LRU page-replacement algorithm may require substantial hardware assistance.

Example:



Thrashing

A process is thrashing if it spending more time in swap-in and swap-out than executing. Thrashing occurs when we increase degree of multiprogramming too much.



Thrashing.

As the degree of multiprogramming increases, CPU utilization also increases, although more slowly, until a maximum is reached. If the degree of multiprogramming is increased even further, thrashing sets in, and CPU utilization drops sharply. At this point, to increase CPU utilization and stop thrashing, we must decrease the degree of multiprogramming.

Locality of Reference

Locality of Reference refers to the tendency of the computer program to access instructions whose addresses are near one another. The property of locality of reference is mainly shown by loops and subroutine calls in a program.

- In case of loops in program control processing unit repeatedly refers to the set of instructions that constitute the loop.
- In case of subroutine calls, every time the set of instructions are fetched from memory.
- References to data items also get localized that means same data item is referenced again and again.

Temporal Locality – Temporal locality means current data or instruction that is being fetched may be needed soon. So we should store that data or instruction in the cache memory so that we can avoid again searching in main memory for the same data.

Spatial Locality – Spatial locality means instruction or data near to the current memory location that is being fetched, may be needed soon in the near future. This is slightly different from the temporal locality. Here we are talking about nearly located memory locations while in temporal locality we were talking about the actual memory location that was being fetched.

Bare Machine

In computer science, bare machine refers to a computer executing instructions directly on logic hardware without an intervening operating system. prior to the development of operating systems, sequential instructions were executed on the computer hardware directly using machine language without any system software layer. This approach is termed the "bare machine" precursor to modern operating [illegible] ed systems and firmware Code runs faster on bare machine but creating software is more expensive for bare machines.

Resident Monitor

In computing, a resident monitor is a type of system software program that was used in many early computers from the 1950s to 1970s. It can be considered a precursor to the operating system. The name is derived from a program which is always present in the computer's memory thus being "resident". Because memory was very limited on these systems the resident monitor was often little more than a stub which would gain control at the end of a job and load a non-resident portion to perform required job cleanup and setup tasks.

On a general-use computer using punched card input, the resident monitor governed the machine before and after each job control card was executed, loaded and interpreted each control card, and acted as a job sequencer for batch processing operations. The functions that the resident monitor could perform were: clearing memory from the last used program (with the exception of itself), loading programs, searching for program data and maintaining standard IO routines in memory.