Operating System

Bachelors in Computer Engineering

Chapter three

Memory Management

OUTLINES:

- Introduction: Storage organization, Memory hierarchy
- Storage allocation, Contiguous verses non-contiguous store allocation, logical and physical memory
- Fragmentation, fixed partition and variable partition for multiprogramming, Logical versus physical address space
- Relocation and Protection
- Memory management with swapping: Memory management with bitmaps and linked list, Memory management without swapping,
- Contiguous-memory allocation: memory protection, memory allocation, Fragmentation (Inter fragmentation and external fragmentation)
- Paging, Structure of page table: Hierarchical page table, Hashed page table, Inverted page table, Shared page table,
- Virtual memory- Introduction, Paging, Page Table, Block mapping, Direct mapping, Translation Look Aside Buffers) Demand paging, Thrashing
- Page replacement, Page replacement algorithms: First-in-first-out, Not recently used, Optimal page replacement, Second chance page replacement, Least Recently used, Clock page replacement, Working set page replacement, WS clock page replacement
- Segmentation, Segmentation with paging
- Coalescing and Compaction

Memory

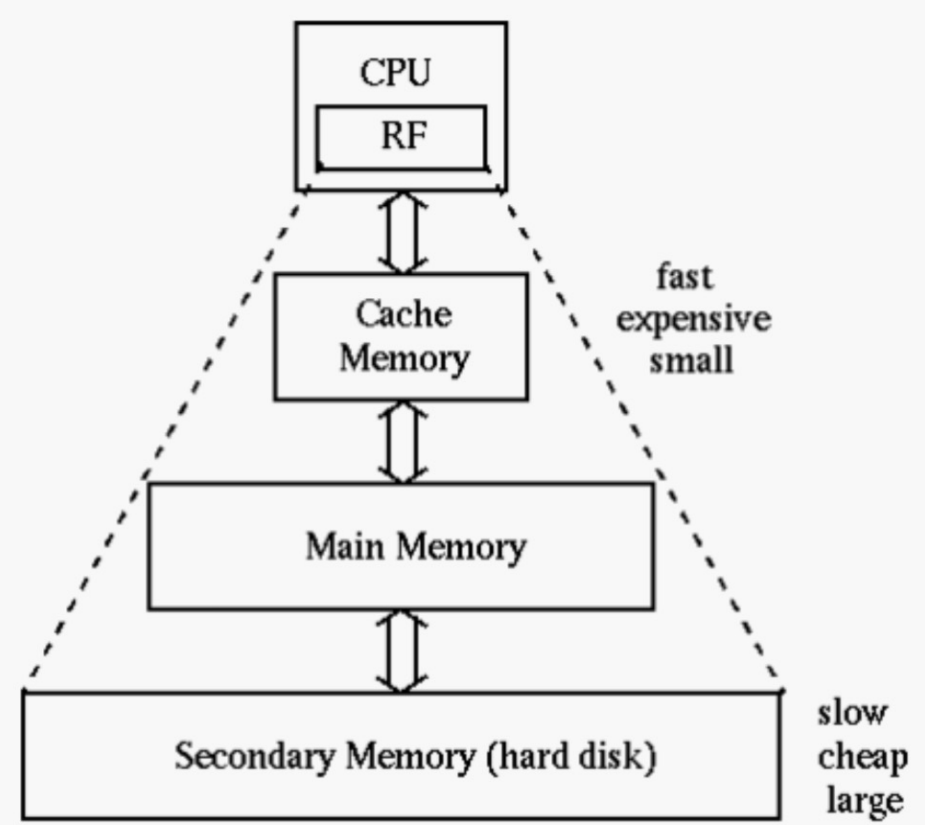
- Memory is the physical device which is used to store programs or data on a temporary or permanent basis for use in a computer or other digital electronics device.
- There are various types of memories in a computer system and are accessed by various processes for their execution.
- Memory management is an important and complex task of an operating system.

Memory Management

- Memory management is the process of controlling and coordinating computer memory.
- It involves assigning blocks of memory to running programs to optimize system performance.
- Memory management is present in hardware, operating systems, and programs/applications.
- In hardware, it includes components like RAM chips, memory caches, and SSDs.
- Operating systems handle memory management by allocating specific memory blocks to programs.
- Memory blocks are adjusted as user demands change.
- At the application level, memory management ensures sufficient memory for running programs.
- Application memory management involves allocation and recycling of memory.
- Allocation ensures adequate memory for objects and data structures.
- Recycling helps free up memory for other programs when it's no longer needed.

Memory Hierarchy

- Memory hierarchy is a ranking of computer memory devices, with devices having the fastest access time at the top of the hierarchy, and devices with slower access times but larger capacity and lower cost at lower levels.



The Memory Hierarhcy

The memories in the hierarchy are as follows:

- At the top level of the memory hierarchy are the CPU's general purpose registers. The registers provide the fastest access to data. The register file is also the smallest memory object in the memory hierarchy.
- The Level One Cache (L1) system is the next highest performance subsystem in the memory hierarchy. The size is usually quite small (typically between 4Kbytes and 32Kbytes), though much larger than the registers available on the CPU chip. Although the Level One Cache size is fixed on the CPU and we cannot expand it, the cost per byte of cache memory is much lower than that of the registers because the cache contains far more storage than is available in all the combined registers.
- The Level Two Cache (L2) is present on some CPUs. The Level Two Cache is generally much larger than the level one cache (e.g., 256 or 512KBytes versus 16 Kilobytes). External caches are actually more expensive than caches that are part of the CPU package.
- Below the Level Two Cache system in the memory hierarchy the main memory subsystem is available. This is the general-purpose, relatively low-cost memory found in most computer systems. Main memory or internal memory is the only one directly accessible to the CPU. The CPU continuously reads instructions stored there and executes them as required.
- Secondary memory also known as external memory or auxiliary storage is different from primary memory in that it is not directly accessible by the CPU. The data stored in such memories are permanently stored and are not lost even the power cut off i.e. non-volatile in nature. Such memories are relatively inexpensive than others.

Fixed sized partition

- The system divides memory into fixed size partition here entire partition is allowed to a process and if there is some wastage inside the partition then it is called internal fragmentation.
- **Advantage:** Management or book keeping is easy.
- **Disadvantage:** Internal fragmentation

Variable size partition

- In the variable size partition, the memory is treated as one unit and space allocated to a process is exactly the same as required and the leftover space can be reused again.

Advantage: There is no internal fragmentation.

Disadvantage: Management is very difficult as memory is becoming purely fragmented after some time.

Memory Management in Monoprogramming

- Monoprogramming allows for the execution of only one process at a time. The entire memory is dedicated to this single process.
- The process in monoprogramming has access to the entire memory space. This enables the process to use the full capacity of the available memory.
- Monoprogramming limits the execution to a single process. No other processes can run concurrently until the current process completes.
- Monoprogramming lacks the capability to execute multiple processes simultaneously. It is not suitable for environments that require multitasking or running multiple programs concurrently.
- Monoprogramming was commonly utilized in early operating systems like MS-DOS. These systems were designed for running a single program at a time.

Memory Management in Multiprogramming

Multiprogramming is required to support multiple users (processes)

- Used in just about all modern OS
- Requires multiple processes to be resident in memory at the same time
- Increases processor utilization – Especially for I/O bound processes
- Able to analyze and predict system performance
- Able to model the relationship between the quantity of memory and CPU utilization

Multiprogramming with Fixed Partitions

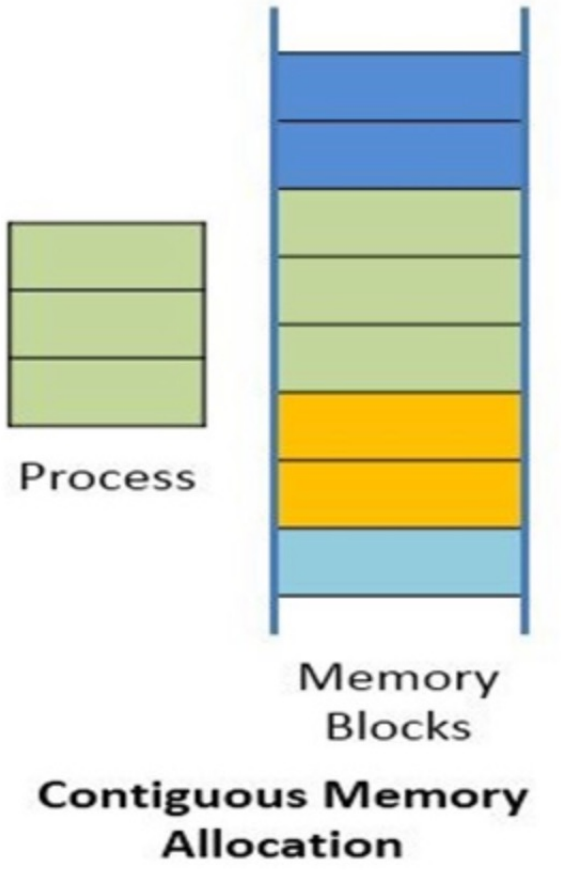
- Desire: Have more than one program in memory at one time
- Solution: Divide memory up into partitions
  - Partitions may be equally or variably sized
- Create an input queue to place processes in the smallest partition that is large enough to hold the process, ensures efficient memory utilization.
- Problems:
  - Some memory partitions may have processes waiting while other partitions are unused
  - Wasted space in larger partitions not used by processes.
- May be used for batch systems, where memory requirements can be modeled
- Not good for interactive systems that often have dynamic memory requirements i.e. require flexibility in allocating memory.

Storage Allocation

- storage allocation refers to the management of memory or disk space for various processes and data.
- There are different types of storage allocation techniques, and they are applied to both primary memory (RAM) and secondary memory (disk storage).
- There are mainly two types of storage allocation:
  - Contiguous and non-contiguous memory allocation

Contiguous Memory Allocation

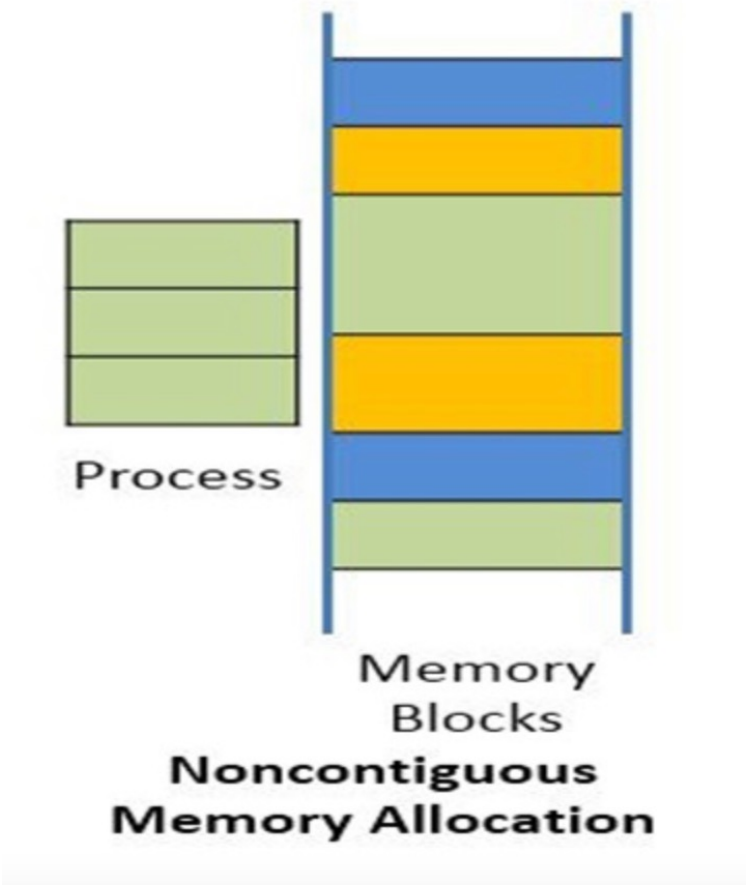
- When a process requests the memory, a single contiguous section of memory blocks is allotted depending on its requirements.
- It is completed by partitioning the memory into fixed-sized partitions and assigning every partition to a single process.
- However, it will limit the degree of multiprogramming to the number of fixed partitions done in memory.
- This allocation also leads to internal fragmentation.
  - For example, suppose a fixed-sized memory block assigned to a process is slightly bigger than its demand.
  - In that case, the remaining memory space in the block is referred to as internal fragmentation.
  - When a process in a partition finishes, the partition becomes accessible for another process to run.
- Contiguous memory allocation speeds up process execution by decreasing address translation overheads.



Contiguous Memory Allocation

Non-contiguous memory allocation

- It allows a process to obtain multiple memory blocks in various locations in memory based on its requirements.
- It also reduces memory wastage cause by internal and external fragmentation because it uses the memory holes created by internal and external fragmentation.
- The two methods for making a process's physical address space non-contiguous are paging and segmentation.
- Non-contiguous memory allocation divides the process into blocks ***(pages or segments)*** that are allocated to different areas of memory space based on memory availability.
- Non-contiguous memory allocation can decrease memory wastage, but it also raises address translation overheads.
- As the process portions are stored in separate locations in memory, the memory execution is slowed because time is consumed in address translation.



Noncontiguous Memory Allocation

Fragmentation:

- It is a technique of partition of memory into different blocks or pages while a process are loaded or removed from the memory.
- The free memory space is broken into little pieces; such types of pieces may or may not be of any use to be allocated individually to any process.
- This may give rise to term memory waste or fragmentation.
- Fragmentation refers to the condition of a disk in which files are divided into pieces scattered around the disk.
- Fragmentation occurs naturally when we use a disk frequently, creating, deleting, and modifying files.
- Fragmentation occurs in a dynamic memory allocation system when many of the free blocks are too small to satisfy any request.
- This is entirely invisible to users, but it can slow down the speed at which data is accessed because the disk must search through different parts of the disk to put together a single file.
- There are two types of fragmentation:
  - External fragmentation
  - Internal fragmentation

External fragmentation:

- It happens when a dynamic memory allocation algorithm allocates some memory and small pieces are left over that cannot be effectively used.
- If too much external fragmentation occurs, the amount of usable memory is drastically reduced.
- Total memory space exists to satisfy a request, but it is not contiguous.

Internal Fragmentation:

- Internal fragmentation is the space wasted inside of allocated memory blocks because of restriction on the allowed sizes of allocated blocks.
- Allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.

Logical versus Physical Address Space:

- An address generated by the CPU is a logical address whereas address actually available on memory unit is a physical address. Logical address is also known a Virtual address.
  - Virtual and physical addresses are the same in compile-time and load-time address-binding schemes.
  - Virtual and physical addresses differ in execution-time address-binding scheme.
- The set of all logical addresses generated by a program is referred to as a logical address space.
- The set of all physical addresses corresponding to these logical addresses is referred to as a physical address space.
- The run-time mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device.
- MMU uses following mechanism to convert virtual address to physical address.
  - The value in the base register is added to every address generated by a user process which is treated as offset at the time it is sent to memory.
    - For example, if the base register value is 10000, then an attempt by the user to use address location 100 will be dynamically reallocated to location 10100.
  - The user program deals with virtual addresses; it never sees the real physical addresses.

Relocation and Protection

- **Relocation** is the process of assigning load addresses to various parts of a program and adjusting the code and data in the program to reflect the assigned addresses.
- A linker usually performs relocation in conjunction with symbol resolution, the process of searching files and libraries to replace symbolic references or names of libraries with actual usable addresses in memory before running a program.
- Relocation is typically done by the linker at link time, but it can also be done at run time by a relocating loader, or by the running program itself.
  - Some architecture avoids relocation entirely by deferring address assignment to run time; this is known as zero address arithmetic.
- Relocation is typically done in two steps:

1. Each object file has various sections like code, data etc. To combine all the objects to a single executable, the linker merges all sections of similar type into a single section of that type. The linker then assigns run time addresses to each section and each symbol. At this point, the code(functions) and data (global variables) will have unique run time addresses.
2. Each section refers to one or more symbols which should be modified so that they point to the correct run time addresses based on information stored in a relocation table in the object file.

- The relocation table is a list of pointers created by the compiler or assembler and stored in the object or executable file.
- Each entry in the table, or "fix up", is a pointer to an address in the object code that must be changed when the loader relocates the program.
- Fix ups are designed to support relocation of the program as a complete unit.
- In some cases, each fix up in the table is itself relative to a base address of zero, so the fix ups themselves must be changed as the loader moves through the table.

- **Memory protection** is a way to control memory access rights on a computer, and is a part of most modern processor architectures and operating systems.
- The main purpose of memory protection is to prevent a process from accessing memory that has not been allocated to it.
- This prevents a bug or malware within a process from affecting other processes, or the operating system itself.
- An attempt to access allocated memory results in a hardware fault, called a segmentation fault or storage violation exception, generally causing abnormal termination of the offending process.
- Memory protection for computer security includes additional techniques such as address space layout randomization and executable space protection.

Memory Management with Swapping:

- Each and every process must be in main memory for execution. However, a process can be transferred temporarily out of memory to a backing storage device for certain time period and then brought back into main memory for continued execution.
- This process of transferring a process from main memory to backing storage and vice-versa is known as **Swapping.**
- Transformation of process from main memory to backing memory is called swapped out and from backing memory to main memory is known as swapped in.
- This scheme allows more processes to be run than can be fit into memory at one time.

For example:

- Let us consider a multiprogramming environment with a round-robin CPU-scheduling algorithm. When a quantum expires, the memory management will start to swap out the process that just finished and to swap another process into the memory space that has been freed. Normally, a process that is swapped out will be swapped back into the same memory space it occupied previously.
- Swapping policy is used for priority-based scheduling algorithms. If a higher-priority process arrives and wants services, the memory manager can swap out the lower-process and then load and execute the higher-priority process. When the higher-priority process finishes, the lower-priority process can be swapped backed in and continued. Sometimes this variant of swapping is called roll out, roll in.
- However, all the process cannot be swapped. If we want to swap a process, we must be sure that the process is completely idle. For example, if a process is waiting for an I/O operation then it cannot be swapped to free up memory.
- Swapping process can be shown in following figure.

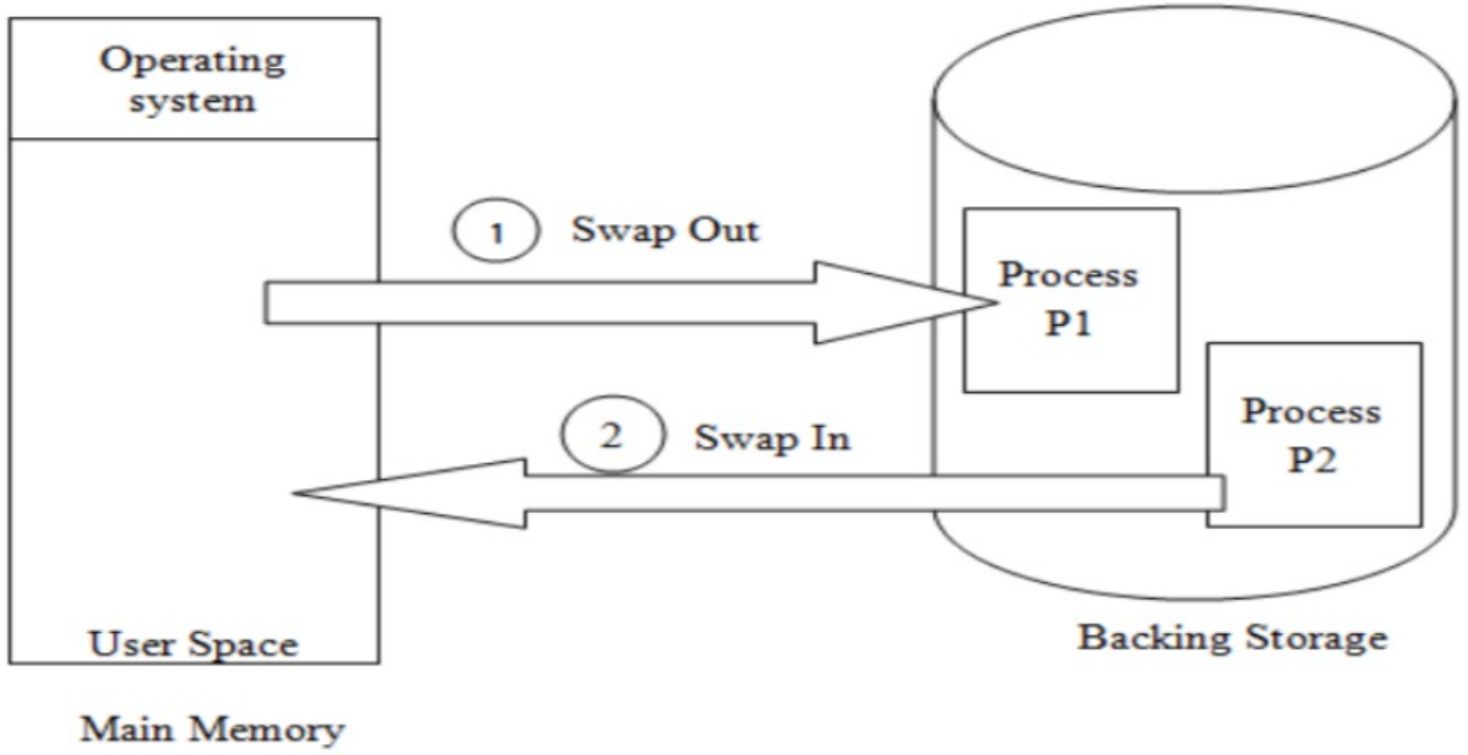
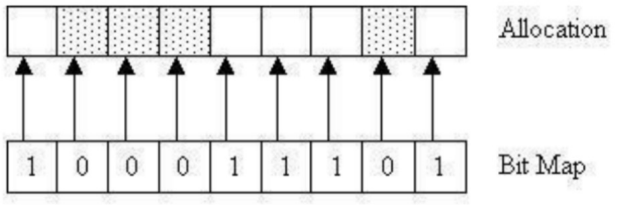


Fig: Swapping of two processes using a disk as a backing storage

Memory Usage with Bit Maps:

- Under this scheme the memory is divided into allocation units and each allocation unit has a corresponding bit in a bit map. If the bit is zero, the memory is free. If the bit in the bit map is one, then the memory is currently being used.
- This scheme can be shown as follows.



- The main decision with this scheme is the size of the allocation unit. The smaller the allocation unit, the larger the bit map has to be. But, if we choose a larger allocation unit, we could waste memory as we may not use all the space allocated in each allocation unit.
- The other problem with a bit map memory scheme is when we need to allocate memory to a process. Assume the allocation size is 4 bytes. If a process requests 256 bytes of memory, we must search the bit map for 64 consecutive zeroes. This is a slow operation and for this reason bit maps are not often used.

Memory Usage with Linked Lists:

- Free and allocated memory can be represented as a linked list. The memory shown above as a bit map can be represented as a linked list as follows.

- Each entry in the list holds the following data
  - P or H: for Process or Hole
  - Starting segment address
  - the length of the memory segment
  - the next pointer is not shown but assumed to be present
- It is possible that two processes can be next to each other and we need to keep them as separate elements in the list so that if one process ends we only return the memory for that process. Consecutive holes, on the other hand, can always be merged into a single list entry. A terminating process can have four combinations of neighbors.
- If X is the terminating process the four combinations are;

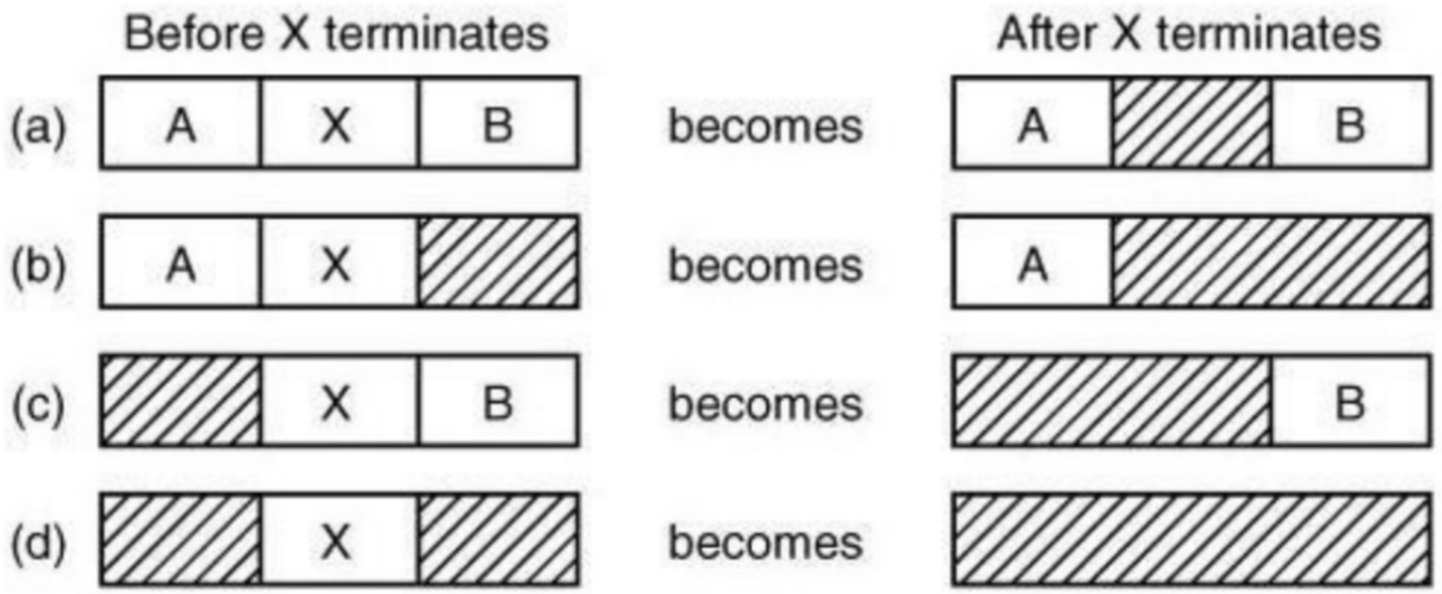


Fig: Four neighbor combinations for the terminating process, X

- In this example, the segment list is kept sorted by address. Sorting this way has the advantage that when a process terminates or is swapped out, updating the list is straightforward. A terminating process normally has two neighbors (except when it is at the very top or very bottom of memory). These may be either processes or holes, leading to the four combinations shown in fig above.
- When the processes and holes are kept on a list sorted by address, several algorithms can be used to allocate memory for a newly created process (or an existing process being swapped in from disk). We assume that the memory manager knows how much memory to allocate.

Coalescing

- In computer science, coalescing is a part of memory management in which two adjacent free blocks of computer memory are merged.
- When a program no longer requires certain blocks of memory, these blocks of memory can be freed.
- Without coalescing, these blocks of memory stay separate from each other in their original requested size, even if they are next to each other.
- If a subsequent request for memory specifies a size of memory that cannot be met with an integer number of these (potentially unequally-sized) freed blocks, these neighboring blocks of freed memory cannot be allocated for this request.
- Coalescing alleviates this issue by setting the neighboring blocks of freed memory to be contiguous without boundaries, such that part or all of it can be allocated for the request.

Compaction:

- Compaction is a process in which the free space is collected in a large memory chunk to make some space available for processes.
- In memory management, swapping creates multiple fragments in the memory because of the processes moving in and out.
- Compaction refers to combining all the empty spaces together and processes.
- Compaction helps to solve the problem of fragmentation, but it requires too much of CPU time.
- It moves all the occupied areas of store to one end and leaves one large free space for incoming jobs, instead of numerous small ones.
- In compaction, the system also maintains relocation information and it must be performed on each new allocation of job to the memory or completion of job from memory.

Virtual Memory

- The main memory is considered as the physical memory in which many programs wants to reside.
- However, the size of such memory is limited and it cannot hold or load all the programs simultaneously.
  - To overcome such types of problem, the overloaded inactive programs are swapped out from the main memory to hard disk for certain period of time. This is done with the help of virtual memory.
- Virtual memory is a memory management technique that allows the execution of processes that may not be completely fit in main memory
  - i.e. it is a technique that temporarily stores inactive parts of the content of RAM on a disk, restoring it to RAM when quick access to it is needed.
- This technique allows running a program whose memory requirement is much higher than available main memory.

- Virtual memory permits software to use additional memory by utilizing the hard disk drive(HDD) as temporary storage.
- This technique involves the manipulation and management of memory by allowing the loading and execution of larger programs or multiple programs simultaneously.
- It is often considered more cost effective than purchasing additional RAM and enables the system to run more programs.
- However the process of mapping data back and forth between the hard drive and the RAM takes longer than accessing it directly from the memory i.e. depending upon virtual memory slows the computer.

Advantages

- Virtual memory allows the execution of programs much larger than the main memory.
- Programmers are practically relieved of the burden of writing the program of limited memory.
  - They are free from worrying about memory requirement and availability.
- It helps to increase the CPU utilization by allowing many users' program to reside in memory.

Paging

- Paging is one of the memory management schemes by which a computer can store and retrieve data from secondary storage for use in main memory.
- It provides and manages processes in memory in term of pages.
- The program to be executed is divided into number of parts called **pages**. The main memory is divided into fixed-sized blocks called **page frames**.
- Pages and page frames are always of equal size.
- Whenever a page frame is available in memory, pages of the process is assigned to it.
  - The operating system maintains a table to keep record of free and occupied page frames.
  - When the program is completed, the page frames occupied by these programs are de-allocated.
- The basic method for implementing paging involves breaking physical memory into frames and logical memory into block of same size called pages.

Structure of Page Table

- A page table is the data structure used by a virtual memory system in a computer operating system to store the mapping between virtual address and physical address
  - i.e. it is used to map virtual page number to physical page number.
- Every **logical address** is divided into **page number (p) and page offset (d)** and every **physical address** is divided into **page frame number (f) and frame offset (d**).
- The virtual page number (p) is used as an index into the page table to find the entry for that virtual page and page offset (d) is combined with base address to define the physical memory address that is sent to the memory unit.
- The content of page table represents the page frame number.
- The page table also indicates the status of each page i.e. whether space in the physical memory is allocated to it or not, if yes which frame is occupied currently.

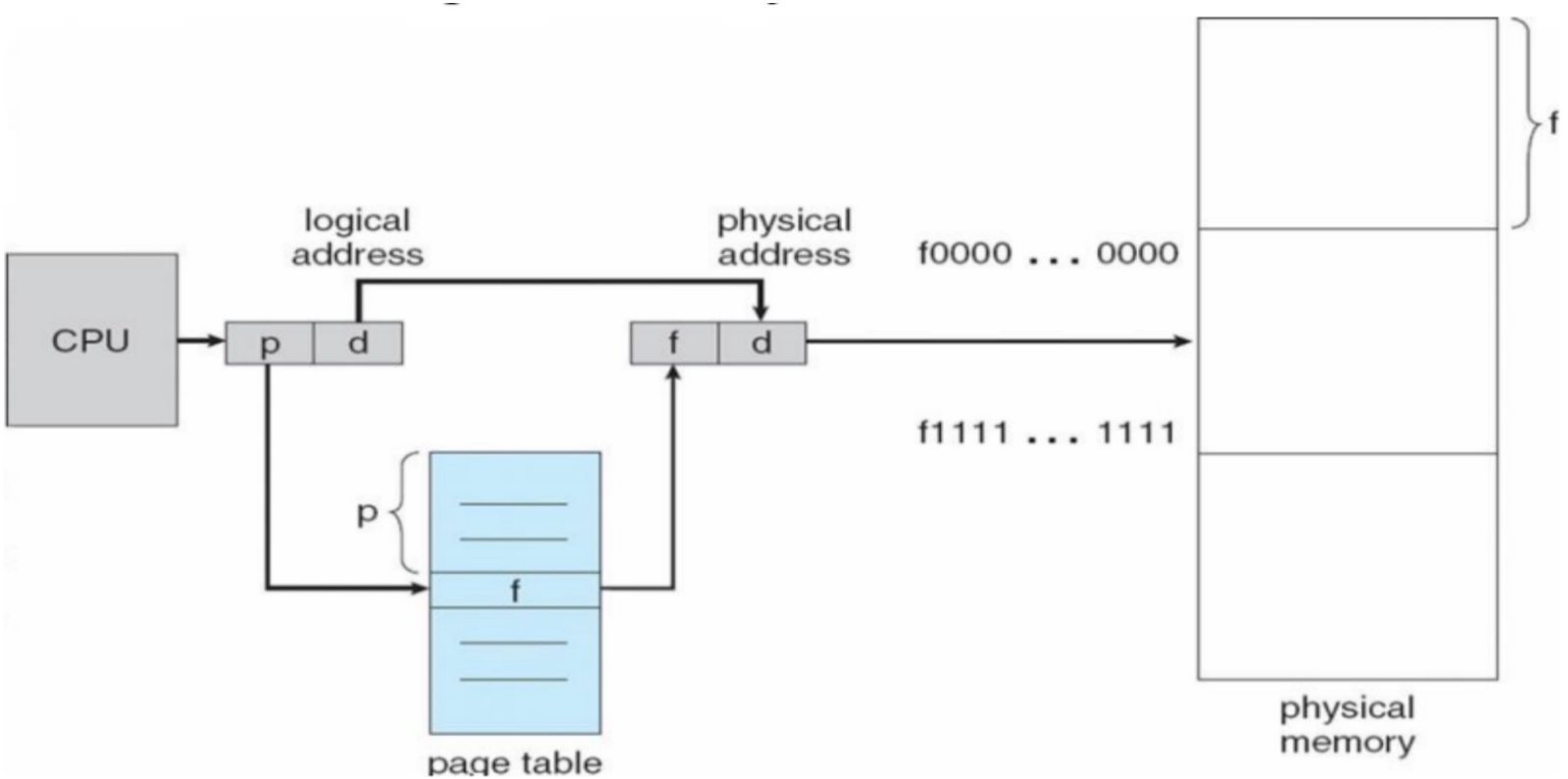


Fig: Paging Hardware

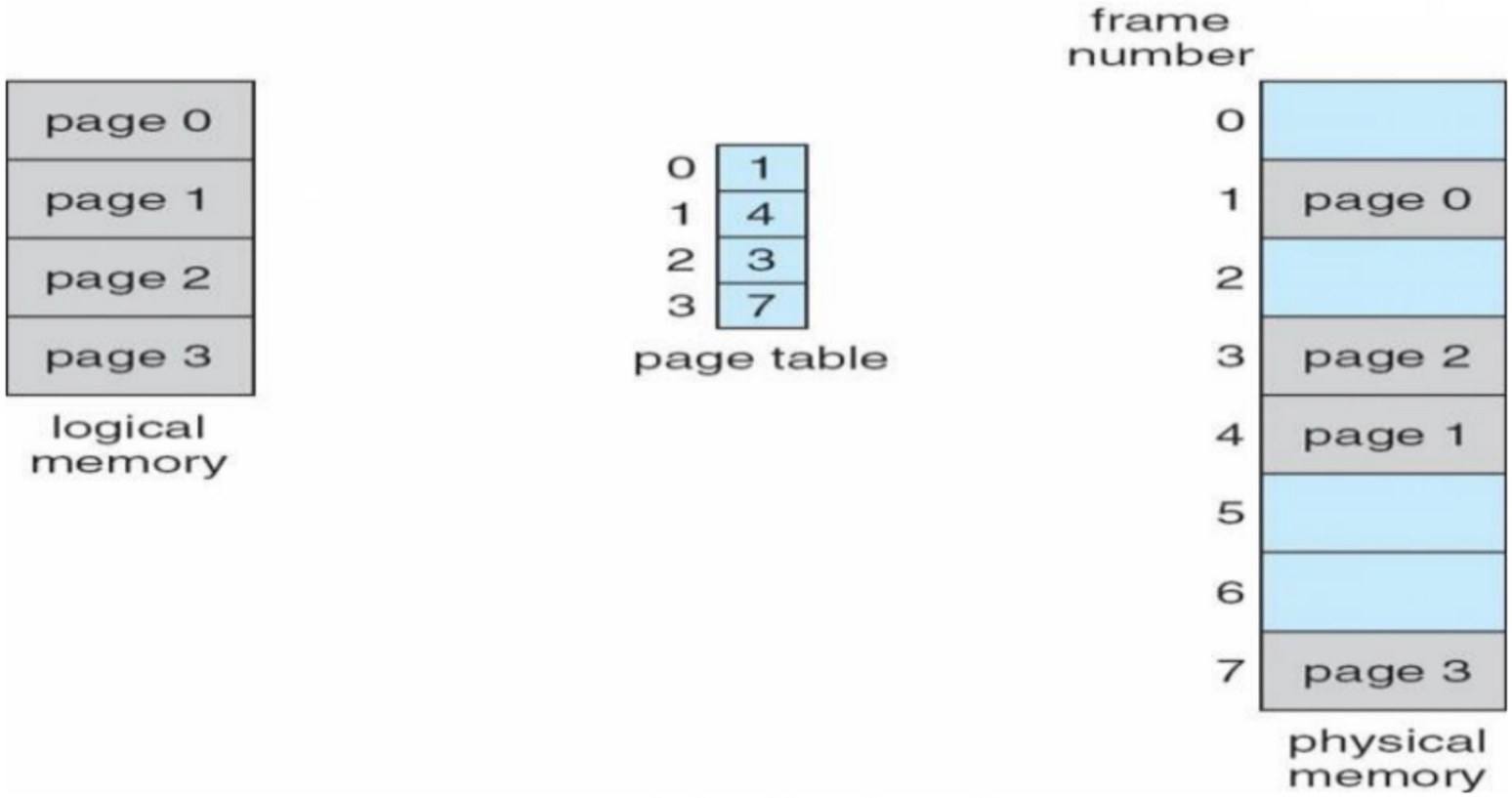


Fig: Paging Concept

- The mapping from virtual address to physical address is done by the mapping table (page table).
- If there are very large virtual address spaces, there would be problem. So, some methods to handle the large virtual address spaces are as follows;
  - Inverted Page Table
  - Hierarchical Page Table
  - Hashed page table
  - Shared page table

- Student Task: Explain these:
  - Inverted Page Table
  - Hierarchical Page Table
  - Hashed page table
  - Shared page table

Page Fault:

- A page fault occurs when a program attempts to access data or code that is in its address space, but is not currently located in the system RAM.
- When a page fault occurs, the operating system searches the free page frame and loads the required page.
- If no page frame is free, the operating system has to choose a page to remove from memory to make room for the page that has to be brought in.
- The page replacement is done by swapping the required pages from backup storage to main memory and vice- versa.

Page Replacement algorithms

- *Page replacement algorithms are used for swapping the page from backup storage to main memory and vice – versa.*
- There are various page replacement algorithms.
  - First In First Out (FIFO)
  - Not recently used,
  - Optimal page replacement,
  - Second chance page replacement,
  - Least Recently used,
  - Clock page replacement,
  - Working set page replacement,
  - WS clock page replacement

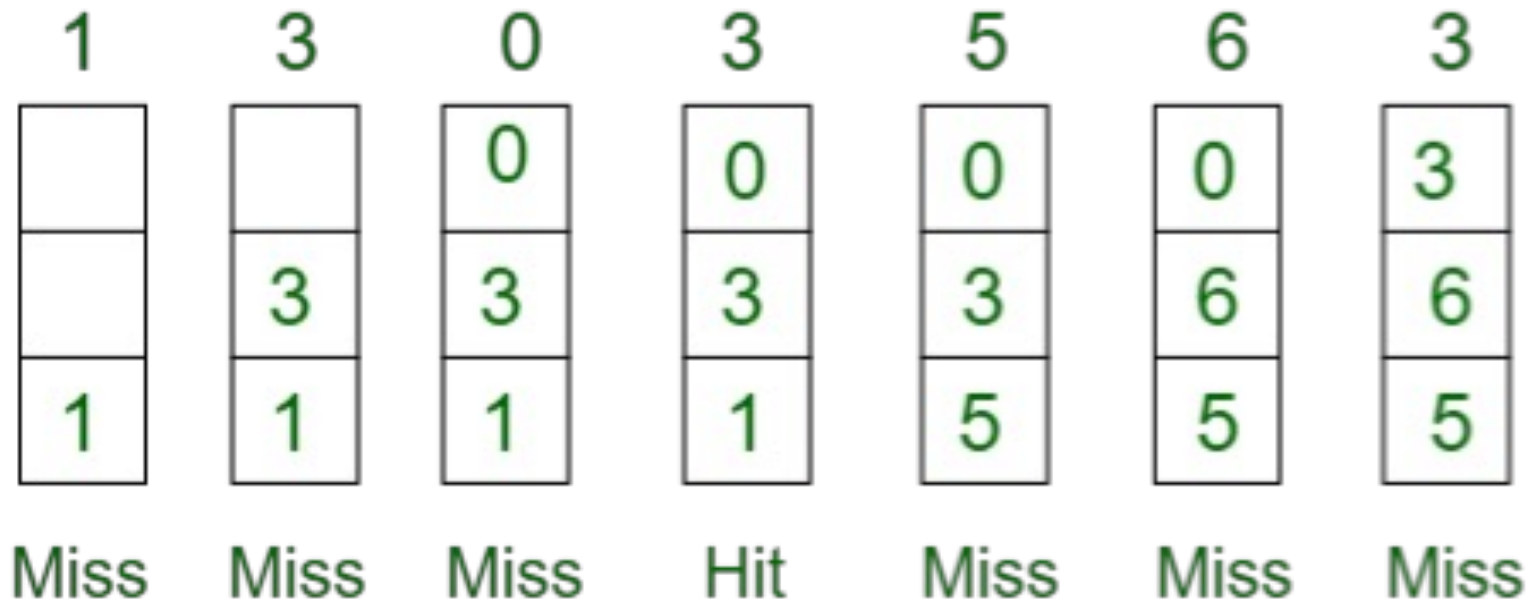
- ***First In First Out (FIFO)***

  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 with 3 page frames. Find number of page faults.

Page reference

1, 3, 0, 3, 5, 6, 3



Total Page Fault = 6

- Initially all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> 3 Page Faults. When 3 comes, it is already in memory so —> 0 Page Faults.
  Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. —>1 Page Fault.
  6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 —>1 Page Fault. Finally when 3 come it is not available so it replaces 0 1 page fault.

- ***Belady's anomaly***

  Belady's anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm. For example, if we consider reference string 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4 and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10 page faults.

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 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, with 4 page frame. Find number of page fault.

Page reference 7,0,1,2,0,3,0,4,2,3,0,3,2,3

No. of Page frame - 4

Total Page Fault = 6

- Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> 4 Page faults 0 is already there so —> 0 Page fault.
- when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future.— >1 Page fault.
- 0 is already there so —> 0 Page fault..
  4 will takes place of 1 —> 1 Page Fault.
  Now for the further page reference string —> 0 Page fault because they are already available in the memory.
- Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

Not Recently Used

- The not recently used (NRU) page replacement algorithm is an algorithm that favors keeping pages in memory that have been recently used. This algorithm works on the following principle: when a page is referenced, a referenced bit is set for that page, marking it as referenced. Similarly, when a page is modified (written to), a modified bit is set. The setting of the bits is usually done by the hardware, although it is possible to do so on the software level as well.

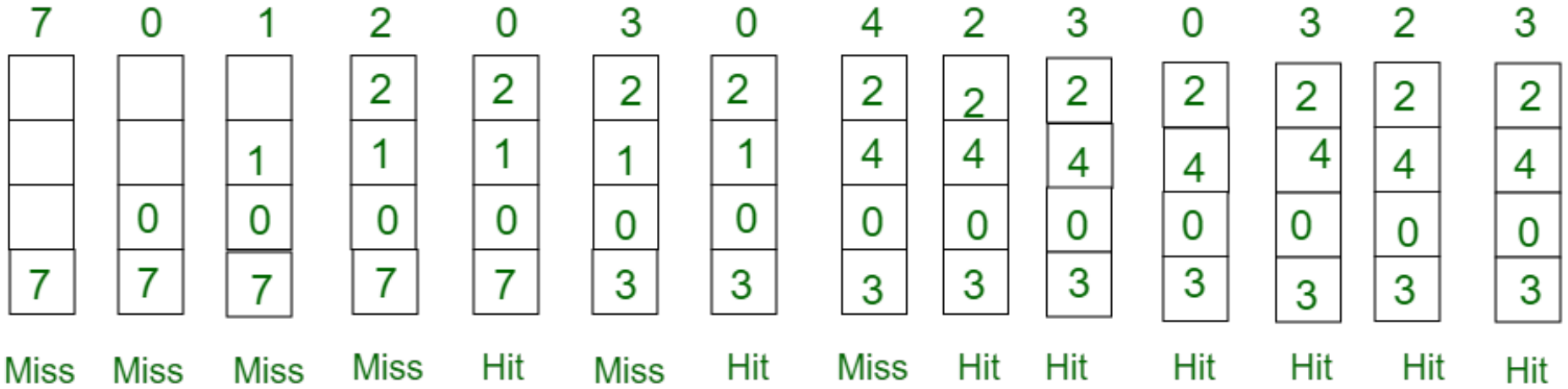
- At a certain fixed time interval, a timer interrupt triggers and clears the referenced bit of all the pages, so only pages referenced within the current timer interval are marked with a referenced bit. When a page needs to be replaced, the operating system divides the pages into four classes:
- 3. referenced, modified
  2. referenced, not modified
  1. Not referenced, modified
  0. Not referenced, not modified

Least Recently Used (LRU) Algorithm

- This algorithm replaces the page that has not been used for the longest period of time.
- It is based on the observation that the pages that have not been used for long time will remain unused and will be replaced.
- In this algorithm, it is necessary to maintain a linked list of all pages in memory, with the most recently used page at the front and the least recently used page at the rear.
- **Example-3:** 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.

Page reference 7,0,1,2,0,3,0,4,2,3,0,3,2,3

No. of Page frame - 4



Total Page Fault = 6

Here LRU has same number of page fault as optimal but it may differ according to question.

Clock Page Replacement Algorithms

- A page replacement algorithm used in operating systems to manage the pages in a page table efficiently.
- It is a simplified version of the Least Recently Used (LRU) algorithm but is more computationally efficient.

1. **Initialization:**
   - Maintain a circular list of pages in memory.
   - Use a reference bit (or use a modified bit) for each page to indicate whether it has been referenced recently.
2. **Page Access:**
   - When a page is accessed, set its reference bit to 1.
3. **Page Replacement:**
   - When a page needs to be replaced, start at the current position of the clock hand (pointing to a page in the circular list).
   - Check the reference bit of the page:
     - If the reference bit is 0, replace the page.
     - If the reference bit is 1, reset the reference bit to 0 and move the clock hand to the next page.
   - Repeat this process until a page is replaced.

- The idea behind the Clock Algorithm is to simulate a clock hand moving around a clock face, pointing to pages in memory.
- The algorithm gives a "second chance" to pages that have been referenced recently before considering them for replacement.
- This algorithm is computationally less expensive than tracking exact access times and provides a good approximation of LRU behavior.
- However, it may not be as accurate as LRU in some scenarios.

Second chance page replacement

- The term "Second Chance" is often used as an alternative name for the Clock Algorithm.
- This reflects the idea that pages are given a "second chance" to avoid replacement if they have been recently accessed.
- The algorithm is called "Second Chance" because, during the replacement process, a page with a reference bit of 1 is given a second chance before being considered for replacement.

Working set page replacement

- The working set is a concept to represent the set of pages that a process is actively using during a specific time interval.
- The working set page replacement algorithm aims to keep the working set of a process in memory to ensure good performance.
- The algorithm monitors the pages accessed by a process and tries to keep the working set in memory.

1. **Monitor Page Access:**
   - Track the pages accessed by each process over a certain time interval.
2. **Maintain Working Set:**
   - Keep the pages in the working set in memory. If a page is not in the working set, it may be a candidate for replacement.
3. **Page Replacement:**
   - When a page needs to be replaced, choose a page that is not in the working set.

- The goal is to avoid excessive page faults and ensure that the pages actively used by a process are kept in memory.

WS clock page replacement

- It is an enhancement of the clock page replacement algorithm, which is a simplified version of the Least Recently Used (LRU) algorithm.
- The clock algorithm maintains a circular list of pages in memory and uses a "hand" that moves around the circle. When a page needs to be replaced, the hand points to the next page in the list.
- The WSClock algorithm introduces the concept of a working set to the clock algorithm.
- It associates a timestamp with each page, representing the time since the page was referenced.
- The algorithm uses this timestamp to decide which pages to keep in memory and which to replace.

- Here are the basic steps:

1.Initialize Timestamps:

- Assign a timestamp to each page indicating when it was last referenced.

2.Page Access:

- When a page is accessed, update its timestamp.

3.Clock Algorithm:

- Use the clock algorithm's circular list with a hand pointing to the next page.
- If the page referenced is within the working set (recently accessed), reset its timestamp and move to the next page.
- If the page is not in the working set, consider it as a candidate for replacement.

4.Page Replacement:

- When a replacement is needed, select the page with the oldest timestamp (least recently used) as the victim.

- The WSClock algorithm dynamically adjusts to the working set of each process, aiming to retain pages actively used while still benefiting from the simplicity of the clock algorithm.

Demand Paging

- There is much less physical memory than virtual memory. So, the operating system must be careful that it does not use the physical memory inefficiently.
- One way to save physical memory is to only load virtual pages that are currently being used by the executing program.
  - For example, a database program may be run to query a database.
  - In this case not the entire database needs to be loaded into memory, just those data records that are being examined.
- This technique of only loading virtual pages into memory as they are accessed is known as demand paging.
- Demand paging is a technique which follows that the pages should only be brought into memory if the executing process demands them.
- This is often referred to as lazy evaluation as only those pages demanded by the process are swapped from secondary storage to main memory.

Thrashing

- Thrashing is a condition in which excessive paging operations are taking place i.e. it refer to any situation in which multiple processes are competing for the same resource, and the excessive swapping back and forth between connections causes a slowdown. This causes the performance of the computer to degrade or collapse. The OS can reduce the effects of thrashing and improve performance by choosing a more suitable replacement strategy for pages.
- Let us take an example of a process that does not have enough number of frames. If the process does not have the number of frames it needs to support pages in active use, it will quickly result page fault. So, the process must replace some active pages with the page that requires a frame. However, since all of its pages are in active use, it must replace a page that will be needed again right away.
- Consequently, it quickly faults again and again. This high paging activity is called Thrashing. Or a process is thrashing if it is spending more time in paging then executing.
- Thrashing is often caused:
  - when the system does not have enough memory,
  - the system swap file is not properly configured, or
  - Too much programs are running on the computer.
- To resolve hard drive thrashing, a user can do any of the below.
  - Increase the amount of RAM in the computer.
  - Decrease the amount of programs being run on the computer.
  - Adjust the size of the swap file.

Thank You