Unit-03

Main Memory Management

Main Memory Management

- During execution, programs and data must be in main memory (at least partially).
- Furthermore, to improve both utilization of the CPU and speed of its response to users, the computer system must keep several processes in memory; that is, we must share memory.
- Since the main memory is usually too small to accommodate all the data and programs permanently, the computer system must provide secondary storage to back up the main memory.
- Memory management is the act of managing computer memory.
- This involves providing ways to allocate portions of memory to programs at request and freeing it for reuse when no longer needed.
- The management of the main memory is critical to the computer system.

Different memory management strategies

➢ Memory management is a critical aspect of computer systems that involves organizing and controlling computer memory, ensuring that processes can allocate and deallocate memory as needed. Various strategies and techniques are employed for memory management in operating systems. Here are some common memory management strategies:

- **Contiguous Memory Allocation:**

→ Contiguous memory allocation is a memory management strategy where each process is allocated a single contiguous block of memory. This means that the entire process, including its code, data, and stack, is loaded into a single, unbroken sequence of memory addresses. There are two main types of contiguous memory allocation.

- **Paging:**

→ Memory is divided into fixed-size blocks called pages, and processes are divided into corresponding fixed-size blocks called page frames. Pages are loaded into frames, allowing for more flexible memory allocation.

→ Reduces external fragmentation compared to contiguous memory allocation.

- **Segmentation:**

→ Memory is divided into logically independent segments such as code, data, and stack. Each segment can grow or shrink dynamically.

→ Helps in organizing programs more logically, but can lead to fragmentation.

- **Virtual Memory:**

→ Allows a process to use more memory than is physically available by using disk space as an extension of RAM.

→ Swapping is a technique where parts of a process are moved between main memory and disk as needed.

- **Demand Paging:**

→ Pages are loaded into memory only when they are demanded during program execution.

→ Reduces the initial loading time of a program.

- **Page Replacement Algorithms:**

→ When a page needs to be brought into memory and there is no free frame, a page replacement algorithm determines which page to remove.

→ Examples include FIFO (First-In-First-Out), LRU (Least Recently Used), and Optimal algorithms.

- **Memory Protection:**

→ Ensures that one process cannot access the memory space of another process, preventing unauthorized interference.

- **Memory Mapping:**

→ Maps files or devices into the memory space, allowing processes to read and write to files as if they were in memory.

- **Buddy System:**

→ Allocates memory in powers of two and keeps track of free and allocated blocks using a binary tree structure.

→ Reduces fragmentation but may lead to internal fragmentation.

- **Garbage Collection:**

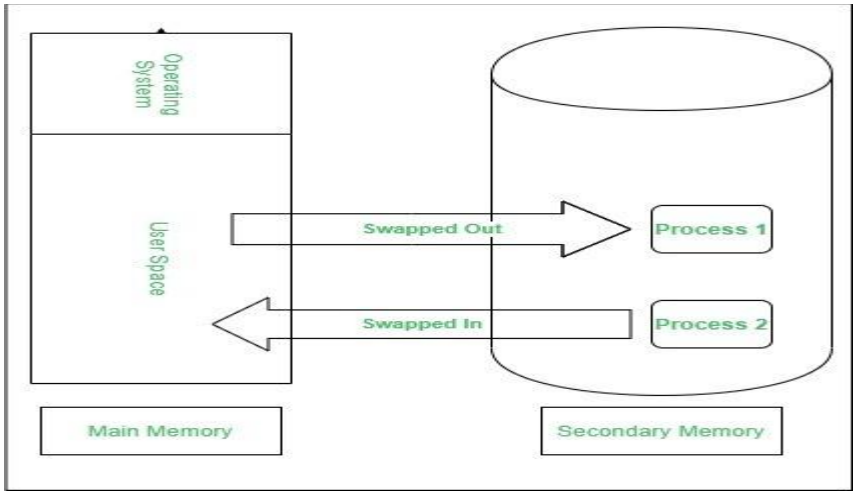
→ Automatically identifies and reclaims memory occupied by objects that are no longer in use.

→ Commonly used in languages with automatic memory management like Java and C#.

Swapping

- Swapping is a memory management technique used in operating systems to cope with the limited physical memory (RAM) available to processes. When a process needs more memory than is currently available in RAM, the operating system can temporarily transfer a part of the process or the entire process to the secondary storage, usually the hard disk. Here are the key concepts associated with swapping.

1. **Swap In:** The method of removing a process from HDD and restoring it to the Main Memory.
2. **Swap Out:** The method of bringing out a process from the main memory and sending it to the HDD so that the processes with higher priority will be executed.



Advantages of Swapping:

- Increased Process Capacity
- Efficient Memory Utilization
- Better System Responsiveness
- Support for Large Programs
- Dynamic Adaptation

Disadvantages of Swapping:

- Performance Overhead
- Thrashing
- Disk Wear and Tear
- Complexity in Management
- Data Integrity Concerns
- Limited Impact on Fragmentation
- Security Risks

Memory allocation strategies

➢ Memory allocation strategies refer to the methods used by computer systems to assign portions of the available memory to different programs or processes. These strategies ensure efficient utilization of memory resources and help manage the allocation and deallocation of memory space. Here are some common memory allocation strategies:

1. **Contiguous Memory Allocation:**

a. Single Partition Allocation:

→ The entire memory is treated as a single block, and only one process is allowed to occupy the entire space.

b. Multiple Partition Allocation:

→ Memory is divided into fixed-size partitions, and each partition can hold one process. Multiple processes can reside in memory simultaneously.

2. **Dynamic Memory Allocation:**

→ Memory is allocated at runtime as needed by a program.

→ Dynamic memory allocation is often implemented using techniques like pointers, malloc(), free(), new, and delete in programming languages like C and C++.

3. **Stack Allocation:**

→ Memory is allocated in a last-in, first-out (LIFO) fashion, resembling a stack data structure.

→ Commonly used for function call management, local variables, and recursive function calls.

4. **Heap Allocation:**

→ Memory is allocated from a heap, and the programmer must explicitly manage the allocation and deallocation of memory.

→ Provides more flexibility but requires careful memory management by the programmer.

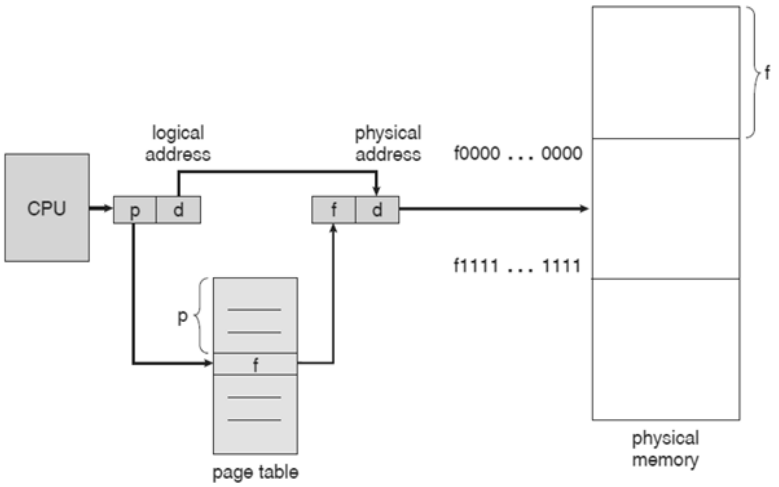
5. **Memory Pool Allocation:**

→ Pre-allocated blocks of memory (pools) are provided to processes, and memory is allocated from these pools.

→ Reduces fragmentation and provides a more controlled allocation process.

Paging and its types

→ Paging is a memory management scheme used by computer operating systems to efficiently manage and allocate memory. In paging, physical memory is divided into fixed-size blocks called "frames," and logical memory is divided into fixed-size blocks called "pages." The size of a page is the same as the size of a frame.



Types of Paging:

1. **Single-Level Paging:**

→ In single-level paging, there is a single page table that contains the mapping between logical pages and physical frames.

→ Simple but can lead to a large page table if the address space is extensive.

2. **Two-Level Paging:**

→ To address the issue of large page tables in single-level paging, a two-level paging scheme is used.

→ The first-level page table indexes into a second-level page table, reducing the overall size of the page table.

3. **Multi-Level Paging (Hierarchical Paging):**

→ Multi-level paging extends the concept of two-level paging to more than two levels, creating a hierarchy of page tables.

→ Provides a more flexible and efficient structure for managing large address spaces.

4. **Inverted Page Tables:**

→ In inverted paging, there is a single page table for the entire system, containing entries for each physical frame.

→ Entries in the table point to the corresponding process and page, allowing for more efficient memory management.

5. **Hashed Page Tables:**

→ Hashed page tables use a hash function to map logical page numbers to frame numbers.

→ This approach can help distribute page table entries more evenly, reducing the size of the page table.

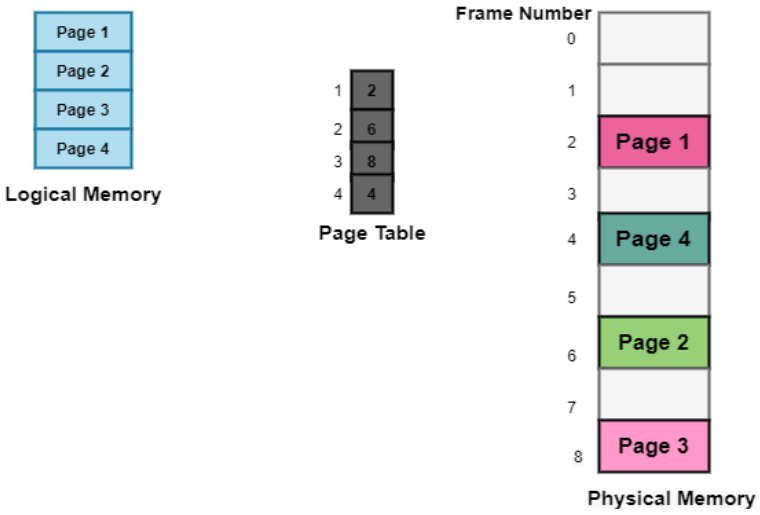
6. **Segmentation with Paging (Paged Segmentation):**

→ Combines segmentation and paging to provide the benefits of both.

→ Logical memory is divided into segments, and each segment is further divided into pages.

Structure of the Page Table

➢ The structure of a page table is a critical component of the paging mechanism in a computer system. The page table is used by the operating system to map logical addresses to physical addresses, allowing processes to access data stored in memory. The specific structure of a page table may vary based on the architecture of the system, but some common components are typically found in page table entries. Here is a general overview of the structure of a page table:



Components of a Page Table Entry:

1. **Page Number (Logical Page Address):**

→ A field representing the logical page number or address. This is the index used to look up the corresponding frame in physical memory.

2. **Frame Number (Physical Frame Address):**

→ A field representing the physical frame number where the corresponding page is stored in physical memory.

3. **Valid/Invalid Bit:**

→ A flag indicating whether the page is currently in physical memory (valid) or not (invalid). When a process tries to access a page, this bit is checked. If the page is not in memory, a page fault occurs, and the page is loaded from secondary storage.

4. **Protection Bits:**

→ Bits indicating the access permissions for the page, such as read-only, read-write, execute, etc. These bits help enforce memory protection.

5. **Dirty Bit:**

→ A flag indicating whether the contents of the page have been modified since it was loaded into memory. This information is useful for implementing copy-on-write strategies and optimizing page replacement.

6. **Reference Bit (Used Bit):**

→ A flag indicating whether the page has been accessed (read or written) recently. This information is used by page replacement algorithms to determine which pages are actively being used.

7. **Additional Control Bits:**

→ Depending on the system architecture and requirements, there may be additional control bits for specific purposes, such as caching information, page locking, or other features.

Some of the common techniques used for structuring the Page tables

1. **Hierarchical Paging**

- It is also called Multilevel Paging and it is a very simple methodology.
- When the page table is too big to fit in a contiguous space then this hierarchical paging system is used with several levels.
- In this, the logical address space is broken up into Multiple page tables.
- Hierarchical paging uses the following two types of page tables:

★ Two-Level Page Table

★ Three-Level Page Table

i. Two Level Page Table

→ On this page table, itself is paged. Therefore, here two-page tables; inner page table and outer page table are present. Example: 32-bit logical address space and a page size of 4 KB is divided into A Page Number consisting of 20 bits and A Page Offset consisting of 12 bits. Since the Page table itself paged, the page number is further divided into, A 10-bit page number and A 10-bit page offset.

ii. Three Level Page Table

→ A two-level paging table is not appropriate for the system with a 64-bit logical address space.

→ Hence, for 64-bit logical address space, Three-Level Page Table is used.

→ In this, divide the outer page table first, and then it will result in a Three-level page table as shown below.

→ Example: 64-bit logical address space and a page size of 4 KB is divided.

2. **Hashed Page Tables**

→ Hashed Page Tables use a hash function to map logical page numbers directly to frame numbers. Instead of maintaining a separate table for each page, a single table is used with a hash function to distribute entries uniformly. This approach aims to minimize the size of the page table and provide efficient access to page table entries.

→ For each element in the hash table, there are three fields available,

- The virtual Page Number (hash value, all bits are not part of the page offset)
- The value of the mapped page frame
- A pointer to the next element in the LinkedList

3. **Inverted Page Tables**

→ Inverted Page Tables represent a different approach to page table structuring. Instead of having a table for each process, a single global table is used, and each entry corresponds to a frame in physical memory. Entries in the table point to the process and page that occupy the respective frame.

→ Each entry in the page table contains the fields such as

- Page number
- Process ID
- Control bits
- Chained Pointer

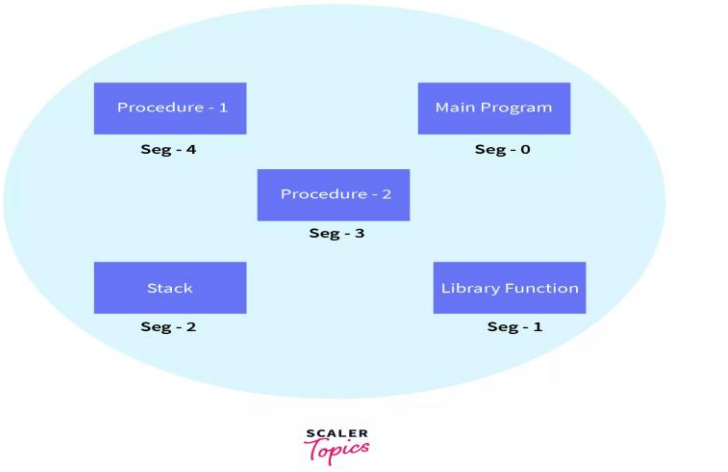
Segmentation

→ In Operating Systems, Segmentation is a memory management technique in which the memory is divided into variable size parts. Each part is known as a segment which can be allocated to a process.

→ Segmentation is the Memory management scheme that supports user view of memory.

→ A segment is a logical unit such as: main program, procedure, function, method, object, local variables, global variables, common block, stack, symbol table, arrays.

→ The details about each segment are stored in a table called a segment table. The segment table is stored in one (or many) of the segments.



→ Segment table contains mainly two information about segment:

- **Base:** It is the base address of the segment. The segment base contains the starting physical address of the segments residing in the memory.
- **Limit:** The segment limit is also known as segment offset. It is the length of the segment.

★ **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.

Advantages of Segmentation

(1) Logical Organization:
(2) Modular Programming
(3) Memory Protection
(4) Dynamic Memory Allocation
(5) Sharing and Reusability
(6) Simplicity in Address Translation
(7) Protection Against Fragmentation
(8) Scalability
(9) Ease of Code Maintenance
(10) Simple Swapping

Disadvantages of Segmentation

(1) Fragmentation
(2) Complex Address Translation
(3) Variable Segment Sizes
(4) Overhead
(5) Limited Scalability

(6) Complicated Memory Protection
(7) Potential for Unused Memory
(8) Limited Support for Shared Memory
(9) Suboptimal for Page Sizes
(10) Increased Page Table Size

Virtual memory segmentation

→ Virtual memory segmentation refers to the use of segmentation in the context of virtual memory management. Virtual memory is a memory management technique that provides an "idealized abstraction of the storage resources that are actually available on a given machine" and "creates the illusion to users of a very large (main) memory."

→ In virtual memory segmentation, the logical address space of a process is divided into segments, each representing a distinct portion of the program's address space. Each segment is treated as an independent unit, and the operating system uses a segment table to map logical addresses to physical addresses.

Importance of virtual memory management

- Virtual memory management is a crucial aspect of modern computer systems, playing a key role in providing abstraction, flexibility, and efficient utilization of memory resources. Here are some important aspects and benefits of virtual memory management:

1. **Abstraction of Physical Memory:**

→ Virtual memory allows programs to operate as if they have a larger amount of contiguous memory than is physically available.
→ It abstracts the physical memory details, providing a logical address space for each process.

2. **Isolation and Protection:**

→ Virtual memory enables the isolation of processes. Each process operates in its own virtual address space, preventing one process from accessing the memory of another.
→ Memory protection mechanisms, such as read-only and no-execute permissions, are enforced to enhance system security and stability.

3. **Demand Paging:**

→ Virtual memory systems often employ demand paging, a technique where only the necessary portions of a program are loaded into physical memory when needed.
→ This minimizes the initial load time of programs and conserves physical memory resources.

4. **Dynamic Memory Allocation:**

→ Virtual memory allows for dynamic memory allocation and deallocation during runtime. Programs can request additional memory as needed, and unused memory can be released.
→ This flexibility supports the efficient use of memory resources.

5. **Optimized Resource Utilization:**

→ Virtual memory management optimizes the use of physical memory by allowing multiple processes to share the same physical memory while maintaining the illusion of separate memory spaces.

→ It helps prevent wastage of memory by loading only the required pages into RAM.

6. Large Address Spaces:

→ Virtual memory provides a large, contiguous address space for each process, regardless of the actual amount of physical memory available.

→ This is essential for running complex applications and handling large datasets.

7. Memory-Mapped Files:

→ Virtual memory management allows memory-mapped files, enabling files to be mapped directly into a process's address space.

→ This provides an efficient mechanism for reading and writing data between files and memory.

8. Improved System Stability:

→ Virtual memory helps enhance system stability by preventing processes from directly interfering with each other's memory space.

→ It enables the operating system to isolate and protect different processes, reducing the likelihood of crashes and conflicts.

9. Swapping and Page Replacement:

→ Virtual memory systems use techniques like swapping and page replacement to efficiently manage memory when the demand for memory exceeds physical capacity.

→ Swapping involves moving entire processes between main memory and secondary storage, while page replacement algorithms decide which pages to keep in physical memory based on access patterns.

Comparison of Paging and Segmentation

Paging	Segmentation
Main memory is partitioned into frames or blocks	Main memory is partationed into segments
The logical address space is divided into pages by MMU	The logical address space is divided into segments as specified by the program
The scheme suffers from internal fragmentation or page breaks	The scheme suffers from External fragmentation
OS maintains a free frame list , so that searching of free frame is not necessary	OS maintains the particulars of available memory
OS maintains a page map table for mapping between frames and pages	OS maintains a segment map table for mapping
This scheme does not support the users view of memory	This scheme support the users view of memory
Processor use the page number and displacement to calculate absolute address (p, d)	Processor use the segment number and displacement to calculate the absolute address (p, d)

Page replacement algorithms

- Page replacement algorithms are used in computer operating systems to manage the contents of the page table when a page fault occurs. A page fault happens when the requested page is not present in the main memory (RAM), and the operating system needs to bring it in from secondary storage (usually a disk). Page replacement algorithms determine which page to evict from the memory to make room for the new page. Here are some common page replacement algorithms:

1. **Optimal Page Replacement (OPT or MIN):**

- This algorithm selects the page that will not be used for the longest period in the future.
- It is an idealized algorithm for measuring the performance of other algorithms, but it's impractical for implementation since it requires knowledge of future page accesses.

2. **FIFO (First-In-First-Out):**

- This is a simple and straightforward page replacement algorithm that replaces the oldest page in the memory.
- It is implemented using a queue data structure, and the page at the front of the queue is the one that has been in memory the longest.

3. **LRU (Least Recently Used):**

- LRU replaces the page that has not been used for the longest period.
- It requires maintaining a record of the order in which pages are accessed, and when a page needs to be replaced, the one that hasn't been used for the longest time is selected.

4. **Clock (or Second Chance):**

- The Clock algorithm approximates LRU and is based on a circular list of pages. A "hand" points to the next page to consider for replacement.
- When a page fault occurs, the algorithm checks if the referenced page has been used (referenced bit). If yes, it clears the bit and moves the hand to the next page; if not, it replaces the page.

5. **LFU (Least Frequently Used):**

- LFU replaces the page that is least frequently used, i.e., the page with the lowest access frequency.
- It requires maintaining a counter for each page that is incremented on each access, and the page with the lowest counter value is selected for replacement.

6. **MFU (Most Frequently Used):**

- Similar to LFU, MFU replaces the page that is most frequently used.
- It requires maintaining a counter for each page, and the page with the highest access frequency is selected for replacement.

7. **Random Replacement:**

- This algorithm randomly selects a page from the memory for replacement. While simple, it may not perform well in comparison to other algorithms, especially in scenarios where patterns of page access exist.

8. **Not Recently Used (NRU):**

- NRU divides pages into four classes based on their referenced and modified bits. It then selects a page for replacement from the lowest non-empty class.

- Classes are typically defined as (0,0), (0,1), (1,0), and (1,1), with (0,0) being the least recently used and unmodified pages.

9. Clock Pro (Enhanced Clock):

- An extension of the Clock algorithm that uses additional information, such as the "age" of a page. Pages are periodically aged, and the algorithm replaces the oldest page with low age.

THE END