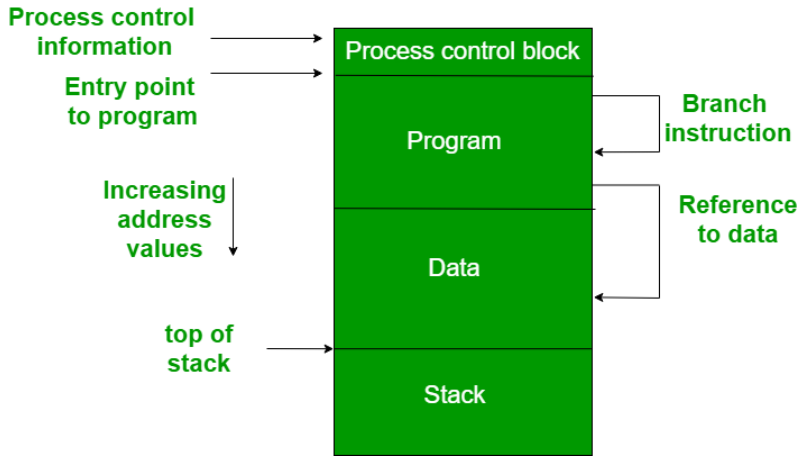
Unit 4

Requirements of Memory Management

Memory management keeps track of the status of each memory location, whether it is allocated or free. It allocates the memory dynamically to the programs at their request and frees it for reuse when it is no longer needed. Memory management meant to satisfy some requirements that we should keep in mind.

These Requirements of memory management are:

1. **Relocation –** The available memory is generally shared among a number of processes in a multiprogramming system, so it is not possible to know in advance which other programs will be resident in main memory at the time of execution of this program. Swapping the active processes in and out of the main memory enables the operating system to have a larger pool of ready-to-execute process .When a program gets swapped out to a disk memory, then it is not always possible that when it is swapped back into main memory then it occupies the previous memory location, since the location may still be occupied by another process. We may need to **relocate** the process to a different area of memory. Thus there is a possibility that program may be moved in main memory due to swapping.



   The figure depicts a process image. The process image is occupying a continuous region of main memory. The operating system will need to know many things including the location of process control information, the execution stack, and the code entry. Within a program, there are memory references in various instructions and these are called logical addresses.

   After loading of the program into main memory, the processor and the operating system must be able to translate logical addresses into physical addresses. Branch instructions contain the address of the next instruction to be executed. Data reference instructions contain the address of byte or word of data referenced.

2. **Protection –** There is always a danger when we have multiple programs at the same time as one program may write to the address space of another program. So every process must be protected against unwanted interference when other process tries to write in a process whether accidental or incidental. Between relocation and protection requirement a trade-off occurs as the satisfaction of relocation requirement increases the difficulty of satisfying the protection requirement. Prediction of the location of a program in main memory is not possible, that's why it is impossible to check the absolute address at compile time to assure protection. Most of the programming language allows the dynamic calculation of address at run time. The memory protection requirement must be satisfied by the processor rather than the operating system because the operating system can hardly control a process when it occupies the processor. Thus it is possible to check the validity of memory references.
3. **Sharing –** A protection mechanism must have to allow several processes to access the same portion of main memory. Allowing each processes access to the same copy of the program rather than have their own separate copy has an advantage. For example, multiple processes may use the same system file and it is natural to load one copy of the file in main memory and let it shared by those processes. It is the task of Memory management to allow controlled access to

the shared areas of memory without compromising the protection. Mechanisms are used to support relocation supported sharing capabilities.

4. **Logical organization –** Main memory is organized as linear or it can be a one-dimensional address space which consists of a sequence of bytes or words. Most of the programs can be organized into modules, some of those are unmodifiable (read-only, execute only) and some of those contain data that can be modified. To effectively deal with a user program, the operating system and computer hardware must support a basic module to provide the required protection and sharing. It has the following advantages:
    - Modules are written and compiled independently and all the references from one module to another module are resolved by `the system at run time.
    - Different modules are provided with different degrees of protection.
    - There are mechanisms by which modules can be shared among processes. Sharing can be provided on a module level that lets the user specify the sharing that is desired.
5. **Physical organization –** The structure of computer memory has two levels referred to as main memory and secondary memory. Main memory is relatively very fast and costly as compared to the secondary memory. Main memory is volatile. Thus secondary memory is provided for storage of data on a long-term basis while the main memory holds currently used programs. The major system concern between main memory and secondary memory is the flow of information and it is impractical for programmers to understand this for two reasons:
    - The programmer may engage in a practice known as overlaying when the main memory available for a program and its data may be insufficient. It allows different modules to be assigned to the same region of memory. One disadvantage is that it is time-consuming for the programmer.
    - In a multiprogramming environment, the programmer does not know how much space will be available at the time of coding and where that space will be located inside the memory.

Fixed (or static) Partitioning in the Operating System

It involves dividing the main memory into a fixed number of partitions at system startup, with each partition being assigned to a process. These partitions remain unchanged throughout the system's operation, providing each process with a designated memory space. This method was widely used in early operating systems and remains relevant in specific contexts like embedded systems and real-time applications. However, while fixed partitioning is simple to implement, it has significant limitations, including inefficiencies caused by internal fragmentation.

1. In fixed partitioning, the memory is divided into fixed-size chunks, with each chunk being reserved for a specific process. When a process requests memory, the operating system assigns it to the appropriate partition. Each partition is of the same size, and the memory allocation is done at system boot time.
2. Fixed partitioning has several advantages over other memory allocation techniques. First, it is simple and easy to implement. Second, it is predictable, meaning the operating system can ensure a minimum amount of memory for each process. Third, it can prevent processes from interfering with each other's memory space, improving the security and stability of the system.
3. However, fixed partitioning also has some disadvantages. It can lead to internal fragmentation, where memory in a partition remains unused. This can happen when the process's memory requirements are smaller than the partition size, leaving some memory unused. Additionally, fixed partitioning limits the number of processes that can run concurrently, as each process requires a dedicated partition.

In operating systems, Memory Management is the function responsible for allocating and managing a computer's main memory. **Memory Management** function keeps track of the status of each memory location, either allocated or free to ensure effective and efficient use of Primary Memory.

There are two Memory Management Techniques:

1. **Contiguous**
2. **Non-Contiguous**

Contiguous Memory Allocation:

In contiguous memory allocation, each process is assigned a single continuous block of memory in the main memory. The entire process is loaded into one contiguous memory region.

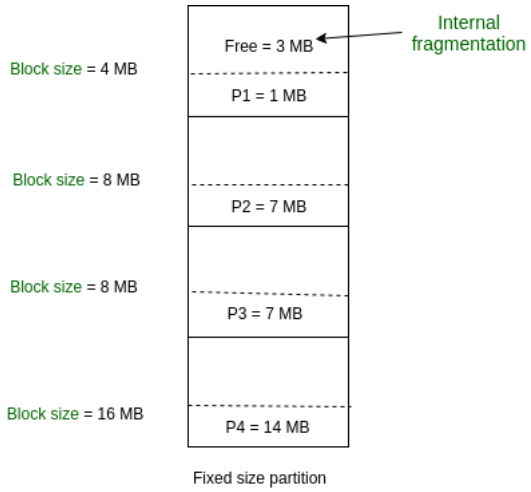
In Contiguous Technique, executing process must be loaded entirely in the main memory.

Contiguous Technique can be divided into:

- Fixed (or static) partitioning
- Variable (or dynamic) partitioning

Fixed Partitioning:

This is the oldest and simplest technique used to put more than one process in the main memory. In this partitioning, the number of partitions (non-overlapping) in **RAM** is **fixed but the size** of each partition may or **may not be the same**. As it is a **contiguous** allocation, hence no spanning is allowed. Here partitions are made before execution or during system configure.



Fixed size partition

Advantages of Fixed Partitioning

- **Easy to implement:** The algorithms required are simple and straightforward.
- **Low overhead:** Requires minimal system resources to manage, ideal for resource-constrained systems.
- **Predictable**: Memory allocation is predictable, with each process receiving a fixed partition.
- **No external fragmentation:** Since the memory is divided into fixed partitions and no spanning is allowed, external fragmentation is avoided.
- **Suitable for systems with a fixed number of processes:** Ideal for systems where the number of processes and their memory requirements are known in advance.
- **Prevents process interference:** Ensures that processes do not interfere with each other's memory, improving system stability.
- **Efficient memory use:** Particularly in systems with fixed, known processes and **batch processing** scenarios.
- **Good for batch processing:** Works well in environments where the number of processes remains constant over time.
- **Better control over memory allocation:** The operating system has clear control over how memory is allocated and managed.
- **Easy to debug:** Fixed Partitioning is easy to debug since the size and location of each process are predetermined.

Disadvantages of Fixed Partitioning

1. **Internal Fragmentation:** Main memory use is inefficient. Any program, no matter how small, occupies an entire partition. This can cause internal fragmentation.
2. **Limit process size:** Process of size greater than the size of the partition in Main Memory cannot be accommodated. The partition size cannot be varied according to the size of the incoming process size. Hence, the process size of 32MB in the above-stated example is invalid.
3. **Limitation on Degree of Multiprogramming:** Partitions in Main Memory are made before execution or during system configure. Main Memory is divided into a fixed number of partitions. Suppose if there are partitions in RAM and are the number of processes, then *n*2<=*n*1 condition must be fulfilled. Number of processes greater than the number of partitions in RAM is invalid in Fixed Partitioning.

   n2<=n1

Clarification:

Internal fragmentation is a notable disadvantage in fixed partitioning, whereas external fragmentation is not applicable because processes cannot span across multiple partitions, and memory is allocated in fixed blocks.

Non-Contiguous Memory Allocation:

In non-contiguous memory allocation, a process is divided into multiple blocks or segments that can be loaded into different parts of the memory, rather than requiring a single continuous block.

Key Features:

Divided memory blocks: A process is divided into smaller chunks (pages, segments) and placed in available memory blocks, which can be located anywhere in the memory.

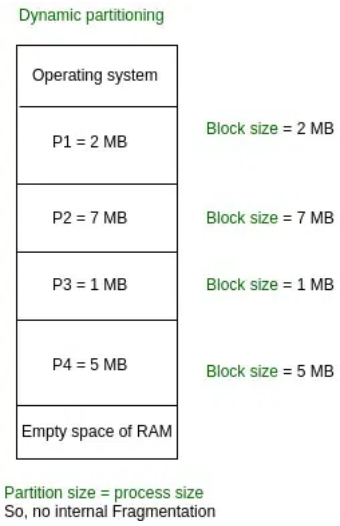
Paging and Segmentation:

- **Paging**: Divides memory into fixed-size blocks called pages. Pages of a process can be placed in any available memory frames.
- **Segmentation:** Divides memory into variable-sized segments based on logical sections of a program, like code, data, and stack.

What is Variable (Dynamic) Partitioning?

It is a part of the Contiguous allocation technique. It is used to alleviate the problem faced by Fixed Partitioning. In contrast with fixed partitioning, partitions are not made before the execution or during system configuration.
Various **features** associated with variable Partitioning-

- Initially, **RAM** is empty and partitions are made during the run-time according to the process's need instead of partitioning during system configuration.
- The size of the partition will be equal to the incoming process.
- The partition size varies according to the need of the process so that internal fragmentation can be avoided to ensure efficient utilization of RAM.
- The number of partitions in RAM is not fixed and depends on the number of incoming processes and the Main Memory's size.



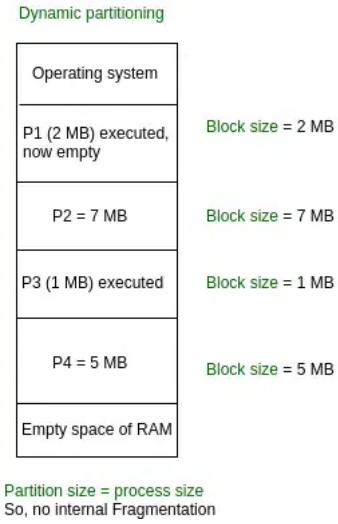
Dynamic Partitioning

Advantages of Variable(Dynamic) Partitioning

- **No Internal Fragmentation:** In variable Partitioning, space in the main memory is allocated strictly according to the need of the process, hence there is no case of **internal fragmentation**. There will be no unused space left in the partition.
- **No restriction on the Degree of Multiprogramming:** More processes can be accommodated due to the absence of internal fragmentation. A process can be loaded until the memory is empty.
- **No Limitation on the Size of the Process:** In Fixed partitioning, the process with a size greater than the size of the largest partition could not be loaded and the process can not be divided as it is invalid in the contiguous allocation technique. Here, In variable partitioning, the process size can't be restricted since the partition size is decided according to the process size.

Disadvantages of Variable(Dynamic) Partitioning

- **Difficult Implementation:** Implementing variable Partitioning is difficult as compared to Fixed Partitioning as it involves the allocation of memory during run-time rather than during system configuration.
- **External Fragmentation:** There will be **external fragmentation** despite the absence of internal fragmentation. For example, suppose in the above example- process P1(2MB) and process P3(1MB) completed their execution. Hence two spaces are left i.e. 2MB and 1MB. Let's suppose process P5 of size 3MB comes. The space in memory cannot be allocated as no spanning is allowed in contiguous allocation. The rule says that the process must be continuously present in the main memory to get executed. Hence it results in External Fragmentation.



No Internal Fragmentation

Now P5 of size 3 MB cannot be accommodated despite the required available space because in contiguous no spanning is allowed.

Key Points On Variable (Dynamic) Partitioning in Operating Systems

- Variable (or dynamic) partitioning is a memory allocation technique that allows memory partitions to be created and resized dynamically as needed.
- The operating system maintains a table of free memory blocks or holes, each of which represents a potential partition. When a process requests memory, the **operating system** searches the table for a suitable hole that can accommodate the requested amount of memory.
- Dynamic partitioning reduces internal fragmentation by allocating memory more efficiently, allows multiple processes to share the same memory space, and is flexible in accommodating processes with varying memory requirements.
- However, dynamic partitioning can also lead to external fragmentation and requires more complex **memory management** algorithms, which can make it slower than fixed partitioning.
- Understanding dynamic partitioning is essential for operating system design and implementation, as well as for system-level programming.

Buddy System

The **Buddy System** is a memory allocation technique covered in operating systems to manage free memory in an efficient and organized manner. Let's explain it in the context of the book "Operating Systems: Internals and Design Principles" by William Stallings.

Buddy System Overview (As per Stallings)

The **Buddy System** is a **dynamic memory allocation scheme** that divides memory into partitions to try to minimize fragmentation. It works as follows:

1. **Memory Block Division**:
   - The entire memory is initially considered a single large block.
   - When a request for memory is made, the system finds the smallest available block that can satisfy the request.
   - If the block is larger than required, it is divided into two equal parts, called "buddies."
   - This process continues until the smallest sufficient block is found.

2. **Buddy Relationship**:
   - Each block of memory has a "buddy" block of the same size. For example:
     - If a block is split into two, each half is considered the buddy of the other.
   - The addresses of the buddies can be determined using binary arithmetic.
3. **Merging Buddies**:
   - When memory is freed, the system checks if its buddy is also free.
   - If both buddies are free, they are merged back into a single block. This process reduces fragmentation and makes larger blocks of memory available.
4. **Advantages**:
   - It reduces external fragmentation by ensuring memory is allocated in power-of-2 sizes.
   - It simplifies the process of merging free memory blocks.
5. **Disadvantages**:
   - It may lead to internal fragmentation since allocations are rounded up to the nearest power of 2.

Example (Based on the Book)

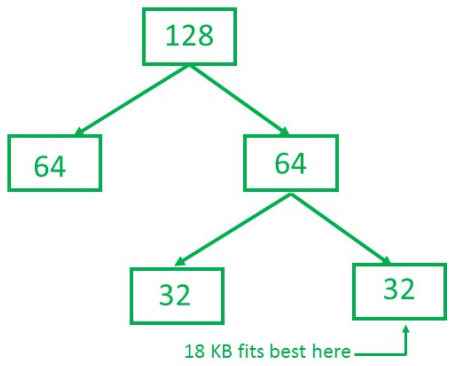
If you have a **1024 KB memory** and a process requests **100 KB**, the Buddy System might work as follows:

1. Start with the entire block of **1024 KB**.
2. The smallest power of 2 that can fit **100 KB** is **128 KB**.
3. The **1024 KB** block is divided:
   - Two **512 KB** buddies.
   - Then one of the **512 KB** blocks is split into two **256 KB** buddies.
   - One of the **256 KB** blocks is split into two **128 KB** buddies.
4. Allocate one **128 KB** block to the process and keep its buddy free.

If the process releases the memory later, the **128 KB** block can be merged with its buddy, and the merging continues up to larger blocks if possible.

Consider a system having buddy system with physical address space 128 KB. Calculate the size of partition for 18 KB process.

Solution:



So, size of partition for 18 KB process = 32 KB. It divides by 2, till possible to get minimum block to fit 18 KB.

Paging in Operating Systems

Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory, thus reducing issues like fragmentation. In *"Operating Systems: Internals and Design Principles"* by William Stallings, the concept of **paging** is covered in detail in the context of virtual memory and memory management techniques.

Overview of Paging

1. **Definition**:

   Paging is a technique where the physical memory is divided into fixed-size blocks called **frames**, and the logical memory (process address space) is divided into blocks of the same size called **pages**.

2. **Purpose**:
   - Avoids fragmentation by allowing processes to be stored non-contiguously in physical memory.
   - Simplifies memory allocation by breaking memory into fixed-size units.

3. **Key Concepts in Paging**:
   - **Page Table**: Each process has a page table that maps logical page numbers to physical frame numbers. Example: Logical address `Page 3` maps to `Frame 7` in physical memory.
   - **Logical vs. Physical Address**:
     - Logical Address: The address generated by the CPU (used by processes).
     - Physical Address: The actual address in the physical memory.

4. **Translation**:

   The Memory Management Unit (MMU) translates logical addresses to physical addresses using the page table.

How Paging Works

1. **Logical Address Breakdown**:
   - The logical address is divided into two parts:
     - **Page Number** (p): Index in the page table.
     - **Page Offset** (d): Displacement within the page.

   Logical Address:

   ```
   Copy code
   Logical Address = [Page Number | Offset]
   ```

2. **Page Table**:
   - The page table contains mappings of logical pages to physical frames.
   - Example:

Logical Page	Physical Frame
Page 0	Frame 4
Page 1	Frame 2

3. **Translation Steps**:
   - The CPU generates a logical address (e.g., Page 1, Offset 200).
   - The MMU looks up Page 1 in the page table and finds it mapped to Frame 2.

- The physical address is computed as:

```
Copy code
Physical Address = (Frame Number × Frame Size) + Offset
```

Advantages of Paging

1. **Eliminates External Fragmentation**: Since memory is allocated in fixed-size frames, no contiguous allocation is required.
2. **Efficient Memory Use**: Non-contiguous allocation allows efficient use of available memory.
3. **Simplifies Allocation**: Fixed-size blocks are easier to manage than variable-size blocks.

Disadvantages of Paging

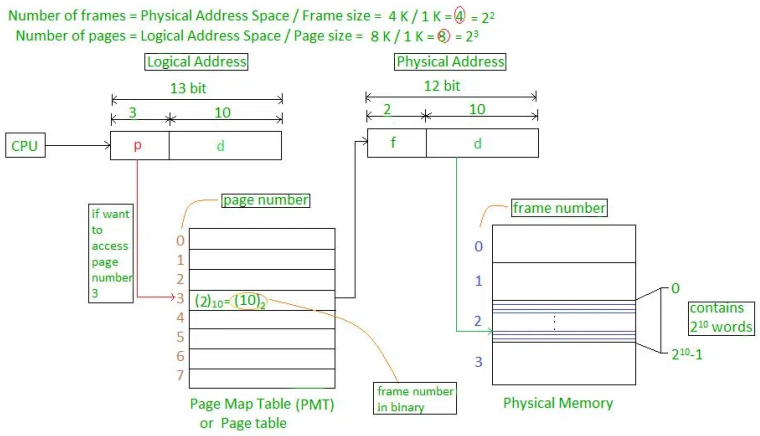
1. **Internal Fragmentation**: If a process doesn't fully use a page, the remaining space in the frame is wasted.
2. **Overhead of Page Tables**: Maintaining large page tables for every process can consume significant memory.
3. **Translation Overhead**: The MMU must translate logical to physical addresses, which adds a slight delay.

Types of Paging

1. **Single-Level Paging**:
   - One page table per process.
   - Simple but inefficient for large address spaces due to the size of page tables.
2. **Multilevel Paging**:
   - Hierarchical structure of page tables (e.g., two-level or three-level paging).
   - Reduces memory overhead by breaking the page table into smaller segments.
3. **Inverted Page Table**:
   - A single table for the entire system instead of per-process tables.
   - Reduces memory overhead but adds lookup complexity.

Example

- Physical Address = 12 bits, then Physical Address Space = 4 K words
- Logical Address = 13 bits, then Logical Address Space = 8 K words
- Page size = frame size = 1 K words (assumption)



Page Table in OS

Page Table is a data structure used by the virtual memory system to store the mapping between logical addresses and physical addresses.

Logical addresses are generated by the CPU for the pages of the processes therefore they are generally used by the processes.

Physical addresses are the actual frame address of the memory. They are generally used by the hardware or more specifically by RAM subsystems.

The image given below considers,

Physical Address Space = M words

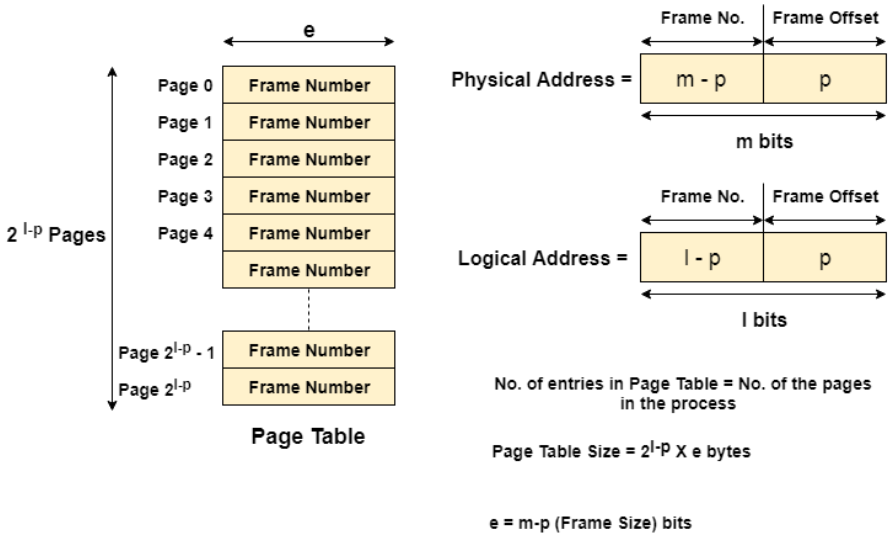
Logical Address Space = L words

Page Size = P words

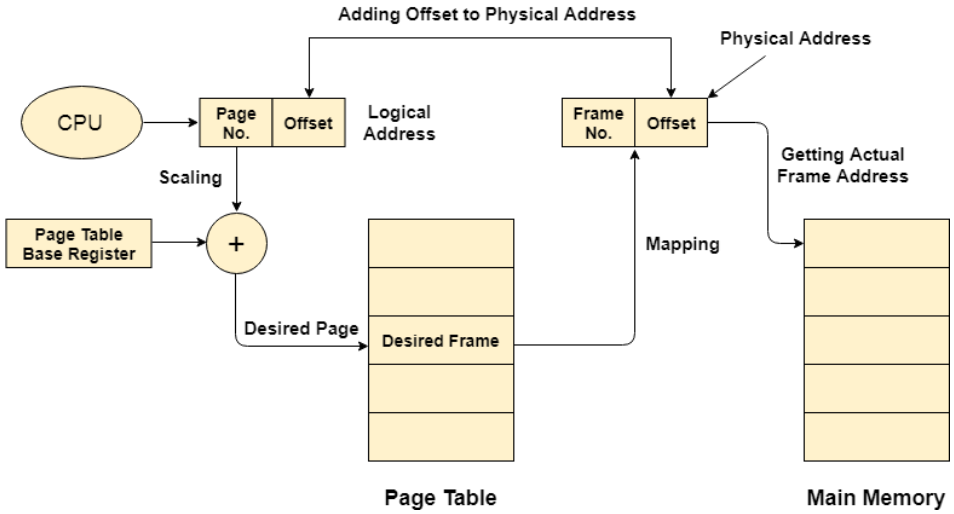
Physical Address = log 2 M = m bits

Logical Address = log 2 L = l bits

page offset = log 2 P = p bits



Memory Management Unit



Segmentation in Operating System

A process is divided into Segments. The chunks that a program is divided into which are not necessarily all of the exact sizes are called segments. Segmentation gives the user's view of the process which paging does not provide. Here the user's view is mapped to physical memory.

Types of Segmentation in Operating Systems

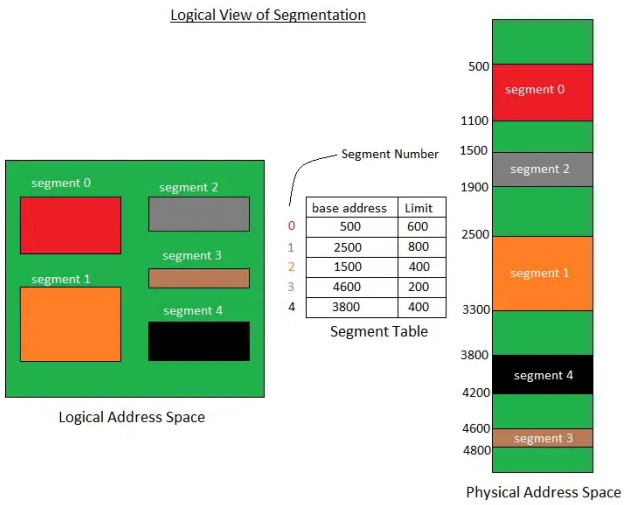
- **Virtual Memory Segmentation:** Each process is divided into a number of segments, but the segmentation is not done all at once. This segmentation may or may not take place at the run time of the program.
- **Simple Segmentation:** Each process is divided into a number of segments, all of which are loaded into memory at run time, though not necessarily contiguously.

There is no simple relationship between logical addresses and physical addresses in segmentation. A table stores the information about all such segments and is called Segment Table.

What is Segment Table?

It maps a two-dimensional Logical address into a one-dimensional Physical address. It's each table entry has:

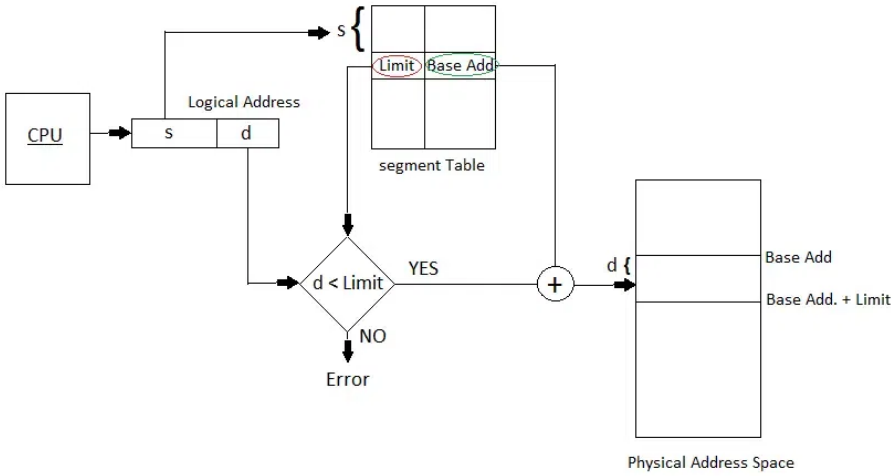
- **Base Address:** It contains the starting physical address where the segments reside in memory.
- **Segment Limit:** Also known as segment offset. It specifies the length of the segment.



Segmentation

Segmentation is crucial in efficient memory management within an operating system.

Translation of Two-dimensional Logical Address to Dimensional Physical Address.



Translation

The address generated by the CPU is divided into:

- **Segment number (s):** Number of bits required to represent the segment.
- **Segment offset (d):** Number of bits required to represent the position of data within a segment.

What is Virtual Memory?

Virtual memory is a memory management technique used by operating systems to give the appearance of a large, continuous block of memory to applications, even if the physical memory (RAM) is limited. It allows the system to compensate for physical memory shortages, enabling larger applications to run on systems with less RAM.

A memory hierarchy, consisting of a computer system's memory and a disk, enables a process to operate with only some portions of its address space in memory. A virtual memory is what its name indicates- it is an illusion of a memory that is larger than the real memory. We refer to the software component of virtual memory as a virtual memory manager. The basis

of virtual memory is the noncontiguous memory allocation model. The virtual memory manager removes some components from memory to make room for other components.

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

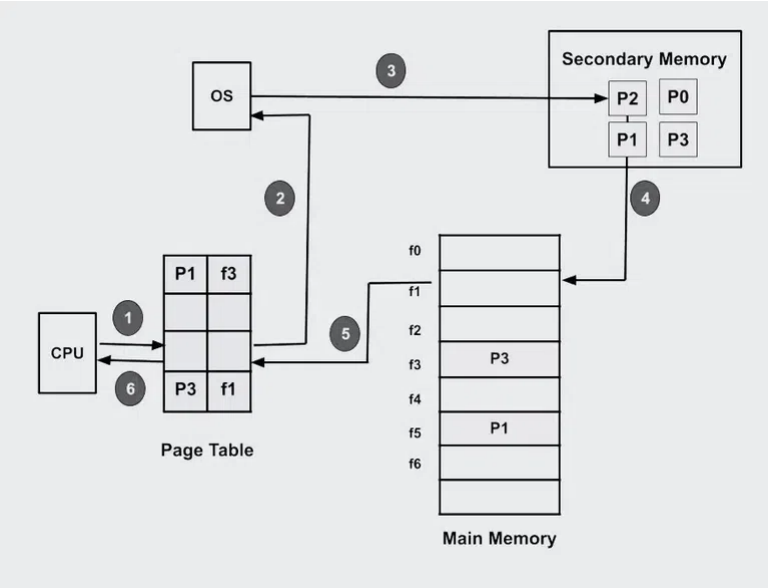
What is Demand Paging?

Demand paging is a technique used in virtual memory systems where pages enter main memory only when requested or needed by the CPU. In demand paging, the operating system loads only the necessary pages of a program into memory at runtime, instead of loading the entire program into memory at the start. A page fault occurred when the program needed to access a page that is not currently in memory.

The operating system then loads the required pages from the disk into memory and updates the page tables accordingly. This process is transparent to the running program and it continues to run as if the page had always been in memory.

Working Process of Demand Paging

Let us understand this with the help of an example. Suppose we want to run a process P which have four pages P0, P1, P2, and P3. Currently, in the page table, we have pages P1 and P3.



The **operating system**'s demand paging mechanism follows a few steps in its operation.

- **Program Execution:** Upon launching a program, the operating system allocates a certain amount of memory to the program and establishes a process for it.
- **Creating Page Tables:** To keep track of which program pages are currently in memory and which are on disk, the operating system makes **page tables** for each process.
- **Handling Page Fault:** When a program tries to access a page that isn't in memory at the moment, a page fault happens. In order to determine whether the necessary page is on disk, the operating system pauses the application and consults the page tables.
- **Page Fetch:** The operating system loads the necessary page into memory by retrieving it from the disk if it is there.
- The page's new location in memory is then reflected in the page table.
- **Resuming The Program:** The operating system picks up where it left off when the necessary pages are loaded into memory.

- **Page Replacement:** If there is not enough free memory to hold all the pages a program needs, the operating system may need to replace one or more pages currently in memory with pages currently in memory. on the disk. The page replacement algorithm used by the operating system determines which pages are selected for replacement.
- **Page Cleanup:** When a process terminates, the operating system frees the memory allocated to the process and cleans up the corresponding entries in the page tables.

How Demand Paging in OS Affects System Performance?

Demand paging can improve system performance by reducing the memory needed for programs and allowing multiple programs to run simultaneously. However, if not implemented properly, it can cause performance issues. When a program needs a part that isn't in the main memory, the operating system must fetch it from the hard disk, which takes time and pauses the program. This can cause delays, and if the system runs out of memory, it will need to frequently swap pages in and out, increasing delays and reducing performance.

Page Replacement

1. FIFO (First-In, First-Out)

How It Works:

- The **oldest page** in memory (the one that was brought in first) is replaced when a page fault occurs.
- Maintains a queue of pages in memory. When a new page is added, it is placed at the back, and the front page is removed if needed.

Example:
Let's assume:

- Page reference string: `7, 0, 1, 2, 0, 3, 0, 4`
- Memory has **3 frames**.

Steps:

1. Start empty: [ ], [7], [7, 0], [7, 0, 1]
2. Reference `2`: Replace `7` → [0, 1, 2]
3. Reference `0`: No change → [0, 1, 2]
4. Reference `3`: Replace `0` → [1, 2, 3]
5. Reference `0`: Replace `1` → [2, 3, 0]
6. Reference `4`: Replace `2` → [3, 0, 4].

Advantages:

- Simple to implement.

Disadvantages:

- **Not always optimal**: May replace a page that will be used again soon (e.g., Belady's Anomaly, where adding more frames increases page faults).

---

2. LRU (Least Recently Used)

How It Works:

- Replaces the **page that has not been used for the longest time**.
- Assumes that pages used recently are more likely to be accessed again soon.
- Maintains a history of page references (e.g., with timestamps or a stack).

Example:
Using the same reference string
`7, 0, 1, 2, 0, 3, 0, 4` and **3 frames**:

Steps:

1. Start empty: [ ], [7], [7, 0], [7, 0, 1]
2. Reference `2`: Replace **7** (least recently used) → [0, 1, 2]
3. Reference `0`: No change → [0, 1, 2]
4. Reference `3`: Replace **1** → [0, 2, 3]
5. Reference `0`: No change → [0, 2, 3]
6. Reference `4`: Replace **2** → [0, 3, 4].

Advantages:

- More accurate than FIFO, as it considers recent usage patterns.

Disadvantages:

- **Implementation complexity**: Requires maintaining a record of access history (e.g., timestamps, counters).

---

3. Optimal Page Replacement

How It Works:

- Replaces the **page that will not be used for the longest period of time in the future**.
- This algorithm is **theoretically optimal** because it minimizes page faults. However, it requires knowledge of the future sequence of memory references (which is impractical in real-world systems).

Example:
Using the same reference string
`7, 0, 1, 2, 0, 3, 0, 4` and **3 frames**:

Steps:

1. Start empty: [ ], [7], [7, 0], [7, 0, 1]
2. Reference `2`: Replace **7** (used farthest in the future) → [0, 1, 2]
3. Reference `0`: No change → [0, 1, 2]
4. Reference `3`: Replace **1** (used farthest in the future) → [0, 2, 3]
5. Reference `0`: No change → [0, 2, 3]
6. Reference `4`: Replace **2** (used farthest in the future) → [0, 3, 4].

Advantages:

- Produces the **minimum number of page faults**.

Disadvantages:

- **Impractical**: Requires predicting the future, which is not possible in real systems.

Allocation of Frames

Frame Allocation refers to the distribution of physical memory frames (blocks of fixed size) to processes in a system. Since each process requires a certain amount of memory, the operating system must determine how many frames to allocate to each process while managing the limited physical memory effectively.

Key Aspects of Frame Allocation

1. **Number of Frames:**
   - The total number of frames is equal to the size of the physical memory divided by the frame size.
   - Example: If physical memory is 1 GB and the frame size is 4 KB, there are 4KB1GB=262,144frames.

$$1\,GB4\,KB = 262,144\,frames\frac{1\,\text{GB}}{4\,\text{KB}} = 262,144\,\text{frames}$$

2. **Types of Frame Allocation Strategies:**
   Frame allocation strategies decide how frames are assigned to processes. These strategies aim to balance performance and prevent memory issues like thrashing.

Frame Allocation Strategies

1. **Equal Allocation**:
   - Each process gets an equal number of frames.
   - Example: If 100 frames are available and there are 5 processes, each process gets 5100=20frames.

$$1005 = 20\,frames\frac{100}{5} = 20\,\text{frames}$$

   - **Disadvantage**: Processes with larger memory needs may not have enough frames, leading to frequent page faults.
2. **Proportional Allocation**:
   - Frames are allocated based on the size of the process.
   - Example: If Process A requires 40% of memory and Process B requires 60%, frames are allocated in the same ratio.

$$Formula: Frames\ for\ Process = Total\ Frames \times Total\ Size\ of\ All\ ProcessesSize\ of\ Process$$

$$Frames\ for\ Process = Total\ Frames \times Size\ of\ ProcessTotal\ Size\ of\ All\ Processes\text{Frames for Process} = \text{T}$$

   - **Advantage**: Processes with larger memory needs get more frames.
3. **Priority Allocation**:
   - Frames are allocated based on the priority of the process. High-priority processes receive more frames, ensuring faster execution.
4. **Global vs. Local Allocation**:
   - **Global Allocation**: Frames can be taken from any process when a page fault occurs.
   - **Local Allocation**: Frames are taken only from the process that caused the page fault, preventing interference between processes.

Thrashing

Thrashing occurs when a system spends more time swapping pages in and out of memory (due to frequent page faults) than executing actual processes. It is a severe performance issue caused by insufficient memory allocation or poor management of frames.

Causes of Thrashing

1. **High Degree of Multiprogramming**:
   - Too many processes are running simultaneously, causing memory contention.
2. **Insufficient Frames**:

- If a process does not have enough frames, it will frequently need to replace pages, leading to excessive page faults.

3. **Working Set Exceeds Memory**:
   - If the total memory demand of all processes exceeds physical memory, thrashing occurs.

Working Set Model and Thrashing Prevention

1. **Working Set**:
   - The working set of a process is the set of pages it needs during a particular time interval.
   - If the working set of all processes exceeds available memory, thrashing occurs.
2. **Working Set Model**:
   - The OS monitors the working set of processes and suspends processes whose working set cannot fit in memory, reducing thrashing.

Effects of Thrashing

1. **CPU Utilization Drops**:
   - The CPU waits for memory pages to be swapped in and out, reducing throughput.
2. **Increased Disk I/O**:
   - Excessive swapping increases disk usage, slowing down the entire system.
3. **Longer Process Execution Times**:
   - Processes take longer to complete as they are repeatedly interrupted by page faults.

Thrashing Prevention Techniques

1. **Use of a Good Page Replacement Algorithm**:
   - Algorithms like **LRU** or **Optimal** minimize unnecessary page replacements.
2. **Working Set Strategy**:
   - Ensure each process has enough frames to hold its working set.
3. **Reducing Degree of Multiprogramming**:
   - Limit the number of active processes so that memory demand does not exceed capacity.
4. **Page Fault Frequency (PFF) Control**:
   - The OS monitors the page fault rate and adjusts frame allocation dynamically:
     - If the fault rate is high, allocate more frames.
     - If the fault rate is low, allocate fewer frames to the process.

Illustrative Example of Thrashing

- **Scenario**:
  - Assume a system with 10 frames and 3 processes.
  - Process A requires 5 frames, Process B requires 4, and Process C requires 4.
  - Total demand: 5+4+4=13 frames, but only 10 frames are available.

    5+4+4=135 + 4 + 4 = 13
- **Result**:

- Pages will constantly be swapped in and out, as no process has enough frames to hold its working set. This leads to thrashing.