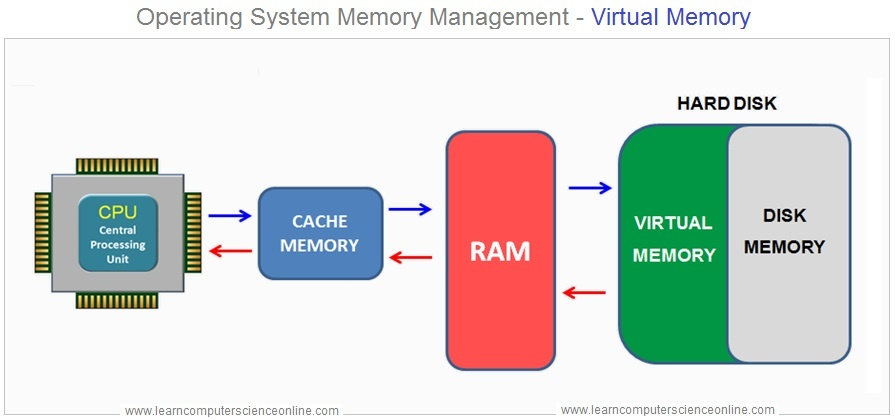
**Introduction to Virtual Memory**

- What is Virtual Memory?

- Definition: Virtual memory is a memory management technique that creates an illusion of a large memory space by using both the RAM and disk storage.

When a computer needs storage for running programs, it uses RAM first.

Virtual memory, which is slower, is used only when the RAM is full.



- Why it's important: Allows execution of processes that exceed the actual physical memory.

| **Feature** | **Virtual Memory** | **Physical Memory (RAM)** |
| --- | --- | --- |
| **Definition** | An abstraction that extends the available memory by using disk storage | The actual hardware (RAM) that stores data and instructions currently being used by the CPU |
| **Location** | On the hard drive or SSD | On the computer’s motherboard |
| **Speed** | Slower (due to disk I/O operations) | Faster (accessed directly by the CPU) |
| **Capacity** | Larger, limited by disk space | Smaller, limited by the amount of RAM installed |
| **Cost** | Lower (cost of additional disk storage) | Higher (cost of RAM modules) |
| **Data Access** | Indirect (via paging and swapping) | Direct (CPU can access data directly) |
| **Volatility** | Non-volatile (data persists on disk) | Volatile (data is lost when power is off) |

- How Virtual Memory Works:

- Paging: Memory is divided into fixed-size pages, and processes are divided into pages that can be loaded into different frames in physical memory.

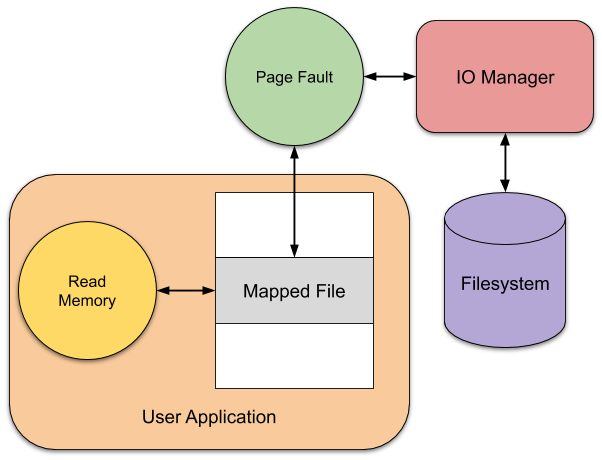
- Page Table: Maintains the mapping between the virtual memory pages and physical memory frames.

- Page Faults:

- Definition: Occurs when a program tries to access a page that is not currently in memory.

- Handling a Page Fault: Steps involved in handling a page fault (interrupt, disk I/O, loading the page into memory).

- Examples of Page Faults: Reading from a virtual memory address that has not been loaded into RAM.

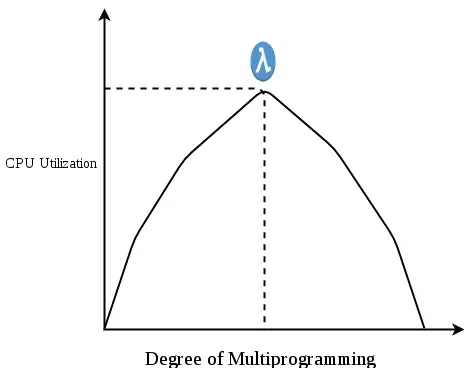


**Thrashing and Introduction to Page Replacement Algorithms**

- Thrashing:

- Definition: A condition where excessive paging occurs, leading to a severe degradation in performance because the system spends more time swapping pages than executing processes.

- Causes: Occurs when the system does not have enough physical memory to handle the working set of processes.

 - Effects: CPU utilization drops, disk I/O increases, and performance collapses.

In the given diagram, the initial degree of multiprogramming up to some extent of point(lambda), the CPU utilization is very high and the system resources are utilized 100%. But if we further increase the degree of multiprogramming the CPU utilization will drastically fall down and the system will spend more time only on the page replacement and the time taken to complete the execution of the process will increase. This situation in the system is called thrashing.

**Page Replacement Algorithms:**

- Purpose: When a page fault occurs and memory is full, the system must decide which page to replace.

- Algorithms:

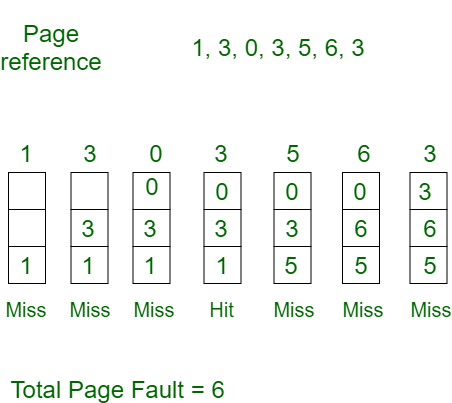
**1. FIFO (First In First Out):**

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.

- Pages are replaced in the order they were loaded.

- Simple but not always efficient.

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

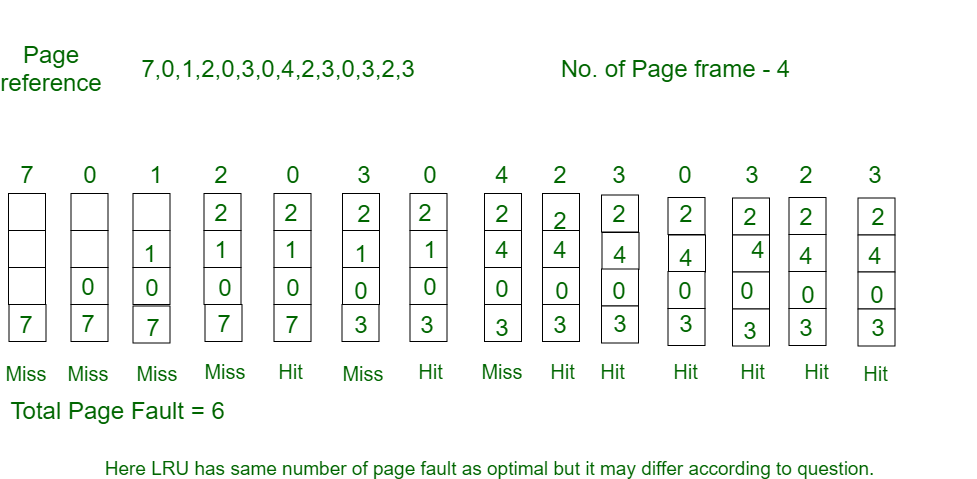


**Belady’s anomaly**proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm.  For example, if we consider reference strings 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4, and 3 slots, we get 9 total page faults,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

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

In this algorithm, page will be replaced which is least recently used.

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



- Replaces the page that has not been used for the longest time.

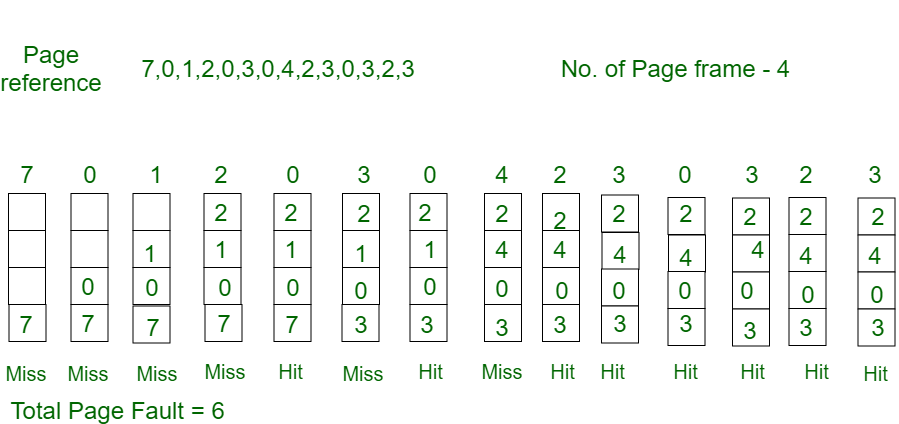
- More effective but requires extra hardware or software for tracking usage.

**3. Optimal Page Replacement:**

- Replaces the page that will not be used for the longest period in the future.

- Theoretical, as future access cannot be known.

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



**React Simulation of Page Replacement Algorithms**

Objective: Build a simple React app to simulate page replacement algorithms (FIFO, LRU, and Optimal).

Key Features of the Simulation:

- Input: Number of frames and a string of page references.

- Output: Visual representation of the memory frames after each page request.

**import React, { useState } from "react";**

**const PageReplacement = () => {**

**const [frames, setFrames] = useState(3); // Number of memory frames**

**const [pages, setPages] = useState(""); // Page reference string**

**const [output, setOutput] = useState([]); // Store output for visualization**

**const [pageHits, setPageHits] = useState(0); // Track page hits**

**const [pageMisses, setPageMisses] = useState(0); // Track page misses**

**// FIFO algorithm**

**const runFIFO = () => {**

**const frameArr = [];**

**const pageArr = pages.split(",").map(Number);**

**const pageFaults = [];**

**let hits = 0;**

**let misses = 0;**

**pageArr.forEach((page) => {**

**if (!frameArr.includes(page)) {**

**if (frameArr.length < frames) {**

**frameArr.push(page);**

**} else {**

**frameArr.shift();**

**frameArr.push(page);**

**}**

**pageFaults.push({ frame: [...frameArr], result: "Miss" });**

**misses++;**

**} else {**

**pageFaults.push({ frame: [...frameArr], result: "Hit" });**

**hits++;**

**}**

**});**

**setOutput(pageFaults);**

**setPageHits(hits);**

**setPageMisses(misses);**

**};**

**// LRU algorithm**

**const runLRU = () => {**

**const frameArr = [];**

**const pageArr = pages.split(",").map(Number);**

**const pageFaults = [];**

**const recentPages = [];**

**let hits = 0;**

**let misses = 0;**

**pageArr.forEach((page) => {**

**if (!frameArr.includes(page)) {**

**if (frameArr.length < frames) {**

**frameArr.push(page);**

**} else {**

**const leastUsed = recentPages.shift(); // Remove least recently used**

**frameArr[frameArr.indexOf(leastUsed)] = page; // Replace it**

**}**

**pageFaults.push({ frame: [...frameArr], result: "Miss" });**

**misses++;**

**} else {**

**recentPages.splice(recentPages.indexOf(page), 1); // Remove the page from recent**

**pageFaults.push({ frame: [...frameArr], result: "Hit" });**

**hits++;**

**}**

**recentPages.push(page); // Add the page as most recently used**

**});**

**setOutput(pageFaults);**

**setPageHits(hits);**

**setPageMisses(misses);**

**};**

**// Optimal algorithm**

**const runOptimal = () => {**

**const frameArr = [];**

**const pageArr = pages.split(",").map(Number);**

**const pageFaults = [];**

**let hits = 0;**

**let misses = 0;**

**pageArr.forEach((page, index) => {**

**if (!frameArr.includes(page)) {**

**if (frameArr.length < frames) {**

**frameArr.push(page);**

**} else {**

**const futurePages = pageArr.slice(index + 1);**

**let farthestPage = -1;**

**let farthestIndex = -1;**

**frameArr.forEach((framePage) => {**

**const futureIndex = futurePages.indexOf(framePage);**

**if (futureIndex === -1) {**

**farthestPage = framePage;**

**farthestIndex = Infinity;**

**} else if (futureIndex > farthestIndex) {**

**farthestPage = framePage;**

**farthestIndex = futureIndex;**

**}**

**});**

**frameArr[frameArr.indexOf(farthestPage)] = page; // Replace the farthest used page**

**}**

**pageFaults.push({ frame: [...frameArr], result: "Miss" });**

**misses++;**

**} else {**

**pageFaults.push({ frame: [...frameArr], result: "Hit" });**

**hits++;**

**}**

**});**

**setOutput(pageFaults);**

**setPageHits(hits);**

**setPageMisses(misses);**

**};**

**return (**

**<div className="bg-gradient-to-br from-indigo-400 to-cyan-400 min-h-screen">**

**<div className="p-6 max-w-md mx-auto bg-gray-100 rounded-lg shadow-md">**

**<h2 className="text-2xl font-bold mb-4 text-center">**

**Page Replacement Simulator**

**</h2>**

**<div className="mb-4">**

**<label className="block text-lg mb-2">Number of Frames:</label>**

**<input**

**type="number"**

**value={frames}**

**onChange={(e) => setFrames(Number(e.target.value))}**

**className="w-full p-2 border rounded-md"**

**/>**

**</div>**

**<div className="mb-4">**

**<label className="block text-lg mb-2">**

**Page Reference String (comma-separated):**

**</label>**

**<input**

**type="text"**

**value={pages}**

**onChange={(e) => setPages(e.target.value)}**

**className="w-full p-2 border rounded-md"**

**/>**

**</div>**

**<div className="flex justify-around mb-6">**

**<button**

**onClick={runFIFO}**

**className="bg-blue-500 text-white px-4 py-2 rounded-md"**

**>**

**Run FIFO**

**</button>**

**<button**

**onClick={runLRU}**

**className="bg-green-500 text-white px-4 py-2 rounded-md"**

**>**

**Run LRU**

**</button>**

**<button**

**onClick={runOptimal}**

**className="bg-purple-500 text-white px-4 py-2 rounded-md"**

**>**

**Run Optimal**

**</button>**

**</div>**

**<h3 className="text-xl font-semibold mb-4">**

**Output (Frames after each page request):**

**</h3>**

**<div className="space-y-2">**

**{output.map((step, index) => (**

**<div key={index} className="p-2 bg-white border rounded-md">**

**<span>{`Step ${index + 1}: [${step.frame.join(", ")}] - `}</span>**

**<span**

**className={**

**step.result === "Miss" ? "text-red-500" : "text-green-500"**

**}**

**>**

**{step.result}**

**</span>{" "}**

**</div>**

**))}**

**</div>**

**<div className="mt-6">**

**<h3 className="text-xl font-semibold">Page Hits and Misses:</h3>**

**<p className="text-lg">Hits: {pageHits}</p>**

**<p className="text-lg">Misses: {pageMisses}</p>**

**</div>**

**</div>**

**</div>**

**);**

**};**

**export default PageReplacement;**