Operating Systems Final Study Guide

Main Memory

Memory Management

Memory management involves managing the main memory to allocate and deallocate spaces for processes and data. Key points include:

- **Responsibilities**:
    - Keep track of memory usage (used/free spaces).
    - Allocate memory efficiently to processes.
    - Ensure protection and isolation of processes.
    - Handle deallocation and compaction to reduce fragmentation.

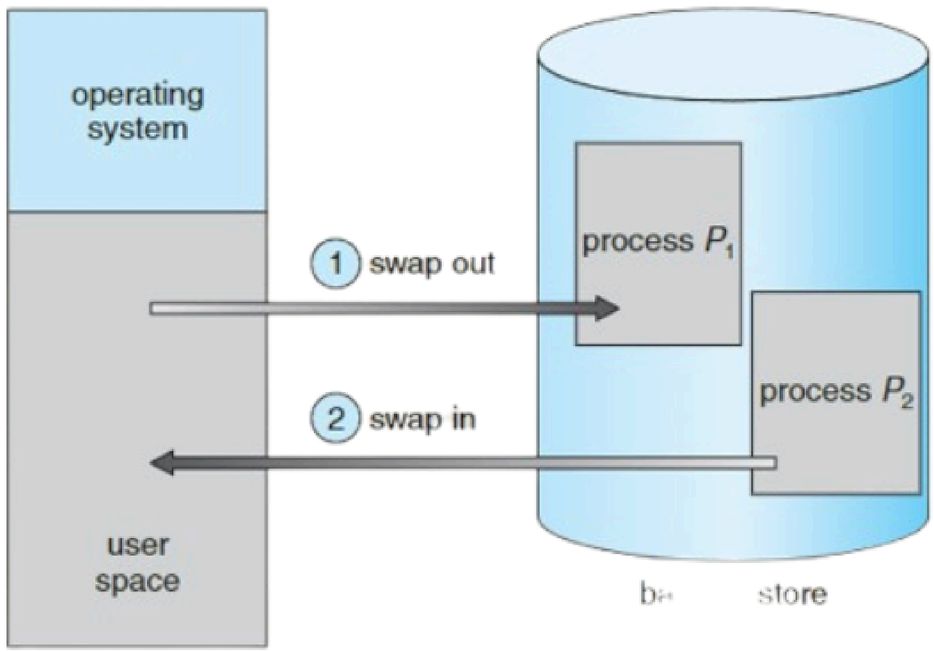
Allocation Strategies (Assignment 10)

- **First-Fit**: Assign the first hole that fits the process.
- **Best-Fit**: Assign the smallest available hole that fits.
- **Worst-Fit**: Assign the largest available hole.

Logical vs. Physical Address Space

- **Logical Address**:
    - Generated by the CPU during program execution.
    - Also called virtual address.
- **Physical Address**:
    - Actual location in memory.
    - Managed by the Memory Management Unit (MMU), which translates logical addresses to physical addresses.
- **Address Binding**:
    - Can occur at compile-time, load-time, or run-time.

Swapping



- **Definition**: A process is temporarily moved out of main memory to secondary storage (disk) to free up space for other processes.
- **Use**: Improves multiprogramming by allowing more processes to run concurrently.
- **Overhead**: Disk I/O operations are slow, so frequent swapping can degrade performance.

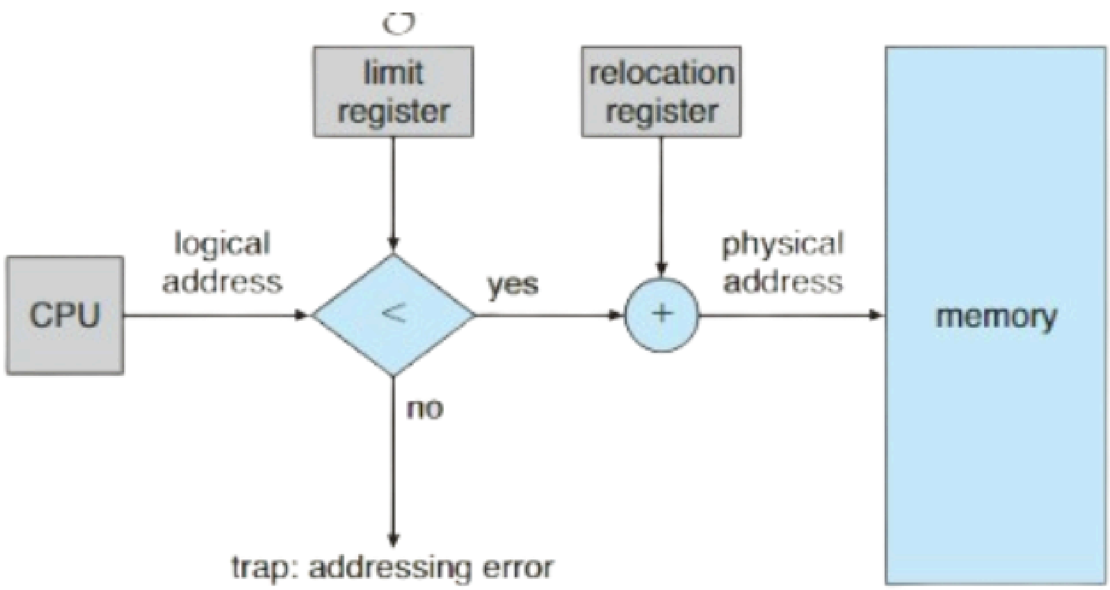
Swapping is efficient only for idle or partially idle processes due to the following reasons:

1. **Minimized Interruptions**:
   - Swapping out an active process disrupts its execution, leading to significant performance degradation.
   - Idle or partially idle processes have minimal CPU or I/O activity, so swapping them out has a lesser impact on the system's responsiveness and throughput.
2. **Avoiding I/O Conflicts**:
   - Active processes often rely on continuous I/O operations. Swapping such processes out can lead to **double buffering**, where the operating system must

temporarily buffer I/O data. This adds overhead and slows down both the swapped process and system performance.

3. **Swapping Overhead**:
    - The time required to swap a process depends on its memory size and disk transfer rate. For an active process, this overhead delays critical operations.
    - Idle processes, on the other hand, do not require immediate CPU or memory access, making the overhead less disruptive.
4. **Resource Optimization**:
    - Swapping out idle processes frees up memory for active processes, improving resource allocation and utilization.
5. **Avoiding Thrashing**:
    - If multiple active processes are swapped in and out frequently, the system may enter a state of **thrashing**, where it spends more time swapping than executing processes. Keeping active processes in memory reduces this risk.

Memory Protection



- Prevents processes from interfering with each other’s memory space.
- **Techniques**:
    - **Base and Limit Registers**: Define the range of accessible addresses.
    - **Segmentation**: Divides memory into segments, each with permissions.
    - **Paging**: Ensures each process can only access its allocated pages.

This diagram explains **Memory Protection** in the context of operating systems using the **limit register** and **relocation register** mechanism. Here's how it works:

Purpose

Memory protection ensures that processes cannot access memory locations outside their allocated range, preventing unintended interference between processes or with the operating system.

Components Explained

1. **Logical Address**:
    - The address generated by the CPU during program execution.
    - It is a relative address that needs to be translated to a physical address.
2. **Relocation Register**:
    - Holds the base address of the process in physical memory.
    - Adds this base address to the logical address to calculate the actual **physical address**.
3. **Limit Register**:
    - Specifies the size of the allocated memory range for the process.
    - Ensures that the process can only access memory within its allocated range.
4. **Memory Access Check**:
    - The CPU compares the logical address to the value in the **limit register**.
    - If the logical address is less than or equal to the limit, the address is valid.
    - If the address is outside the limit, the system triggers a **trap** (an error or interrupt) to handle the invalid access.

Process during Context Switch

1. During a **context switch**:
    - The operating system (OS) dispatcher sets the **relocation register** and **limit register** values for the next process.
    - This ensures that each process operates within its memory bounds and cannot access another process's memory space.
2. After the registers are set:
    - Any memory access performed by the process will automatically be checked against these registers by the hardware.

Why It’s Important

1. **Security**: Prevents processes from accessing memory that they don’t own, which could lead to data corruption or breaches.
2. **Isolation**: Ensures that processes are isolated from each other.
3. **Fault Tolerance**: If a process tries to access an invalid address, the system can safely trap the error and handle it without crashing.

Let me know if you need further clarification!

Fragmentation

- **Definition**: Wasted memory space due to allocation inefficiencies.

Internal Fragmentation

What it is:

- Memory is allocated in fixed-size blocks, but processes often don’t use the entire block, leaving some space unused within the allocated memory.

Example:

- Fixed block size: 4KB.
- Process request: 3.7KB.
- Wasted memory: 0.3KB.

Key Issue:

- Unused memory within allocated blocks adds up across multiple processes.

External Fragmentation

What it is:

- Free memory is available, but it's broken into non-contiguous chunks. As a result, a process requiring a large block of memory can't be allocated even though enough total memory exists.

Example:

- Available memory blocks: **100KB, 150KB, 200KB** (non-contiguous).

- Process request: **250KB**.
- Result: The process **cannot be allocated**, even though the total free memory is **450KB**, because no single block is large enough.

Key Issue:

- Memory gets wasted because free blocks are scattered.

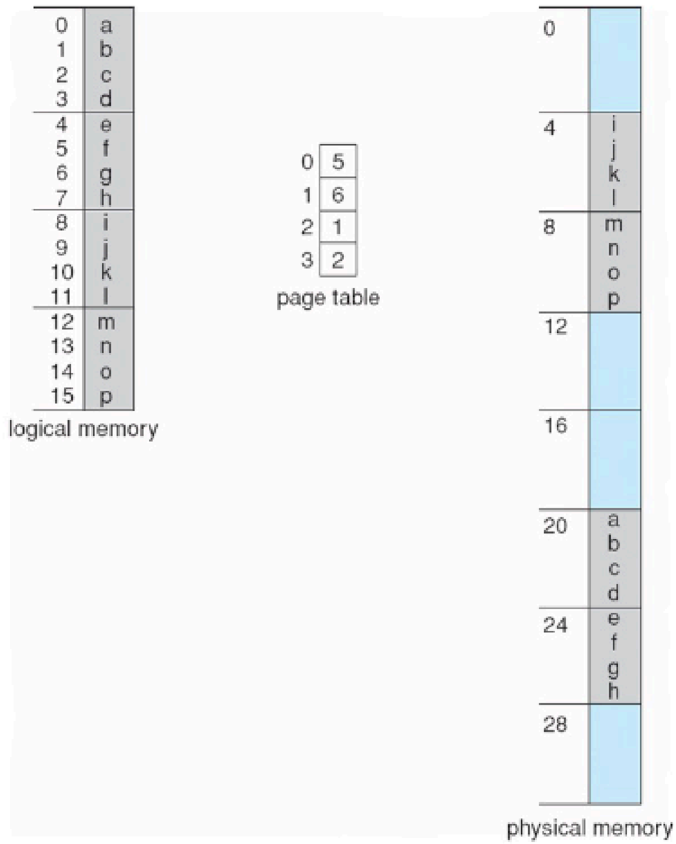
50 Percent Rule:

- For every `N` allocated blocks, approximately `0.5N` blocks are lost to fragmentation.

Paging

- Divides memory into fixed-size blocks called **pages**.
- Eliminates external fragmentation but can cause internal fragmentation.

Page Tables (Assignment 11)



n=2 and *m*=4   32-byte memory and 4-byte pages

- Used to map logical pages to physical frames in memory.
- **Structure**:
    - Each process has its own page table.
    - Contains entries with frame numbers and optional flags (e.g., valid, dirty).

Addressing for Page Tables

- **Address Translation**:
    - Logical address = Page Number + Offset.
    - Page number indexes the page table to find the frame number.
    - Offset determines the specific address within the frame.
- **Multi-Level Page Tables**:
    - Breaks large page tables into smaller, hierarchical tables.
    - Reduces memory overhead for the page table itself.
- **Translation Lookaside Buffer (TLB)**:
    - Cache for page table entries to speed up address translation.

Virtual Memory

Here's a breakdown for **Virtual Memory** concepts with explanations, key points, and potential questions for each section:

Virtual Memory

Shared Pages

- **What it is**:
    - Multiple processes can share common code or data pages in memory, such as shared libraries.
- **How it works**:
    - Pages are mapped into the virtual address space of each process.
    - Changes in shared pages (if allowed) are reflected across all processes.
- **Benefits**:
    - Reduces memory usage by avoiding duplication of common code.
    - Simplifies inter-process communication by sharing data.

Sample Question:

- Explain how shared pages reduce memory usage. Provide an example where two processes share a library.

Demand Paging

- **What it is**:
    - A lazy loading mechanism where pages are loaded into memory only when referenced, not at process start.
- **How it works**:
    - If a required page is not in memory (a **page fault** occurs), the operating system fetches it from disk.
- **Advantages**:

  - Saves memory space by avoiding preloading unused pages.
- **Disadvantages**:
  - Initial page faults can cause performance delays.

Sample Question:

- Describe how demand paging works and explain the role of page faults.

Effective Access Time (EAT)

- **What it is**:
  - The average time required to access memory, considering page faults and TLB (Translation Lookaside Buffer) hits/misses.
- **Formula**: [\text{EAT} = (1 - p) \times \text{Memory Access Time} + p \times (\text{Page Fault Service Time})]
  - p = Probability of a page fault.
  - **Page Fault Service Time** includes disk access and possibly replacing a page.

Sample Question:

- Calculate EAT for a system with:
  - Memory access time = 100ns.
  - Page fault rate = 0.01.
  - Page fault service time = 10ms.

Page Replacement

- **What it is**:
  - When a page fault occurs and memory is full, the system replaces an existing page with the new page.
- **Goal**:
  - Minimize the number of page faults by replacing pages intelligently.

Algorithms (Assignment 12)

1. **FIFO (First-In-First-Out)**:
   - Oldest page is replaced.
2. **Optimal**:

- Replaces the page that will not be used for the longest time (requires future knowledge).
3. **LRU (Least Recently Used)**:
   - Replaces the page that hasn't been used for the longest time.
4. **Second Chance**:
   - Similar to FIFO but gives a "second chance" to pages if they are referenced.

Sample Question:

- Given a reference string and a frame size, calculate page faults for FIFO, LRU, and Optimal algorithms.

Local vs. Global

- **Local Replacement**:
  - Each process has its own frame allocation and replaces pages within its own frames.
- **Global Replacement**:
  - Pages are replaced from a common pool of frames shared by all processes.
- **Comparison**:
  - Global replacement offers flexibility but can cause starvation for some processes.

Sample Question:

- Compare local and global page replacement strategies. Which is better in a highly loaded system?

Thrashing

- **What it is**:
  - When a process spends more time handling page faults than executing, causing a drastic performance drop.
- **Cause**:
  - Occurs when the process's working set (set of pages it actively uses) exceeds the available memory.
- **Solution**:
  - Increase memory, or reduce the degree of multiprogramming.

Sample Question:

- Explain what thrashing is and discuss two ways to mitigate it.

Working Set

- **What it is**:
    - The set of pages a process actively uses over a time window.
- **Purpose**:
    - Helps estimate the memory demand of a process.
- **Working Set Model**:
    - Tracks recently used pages and ensures the working set is in memory to avoid thrashing.

Sample Question:

- Describe the working set model and how it prevents thrashing.

Kernel Memory

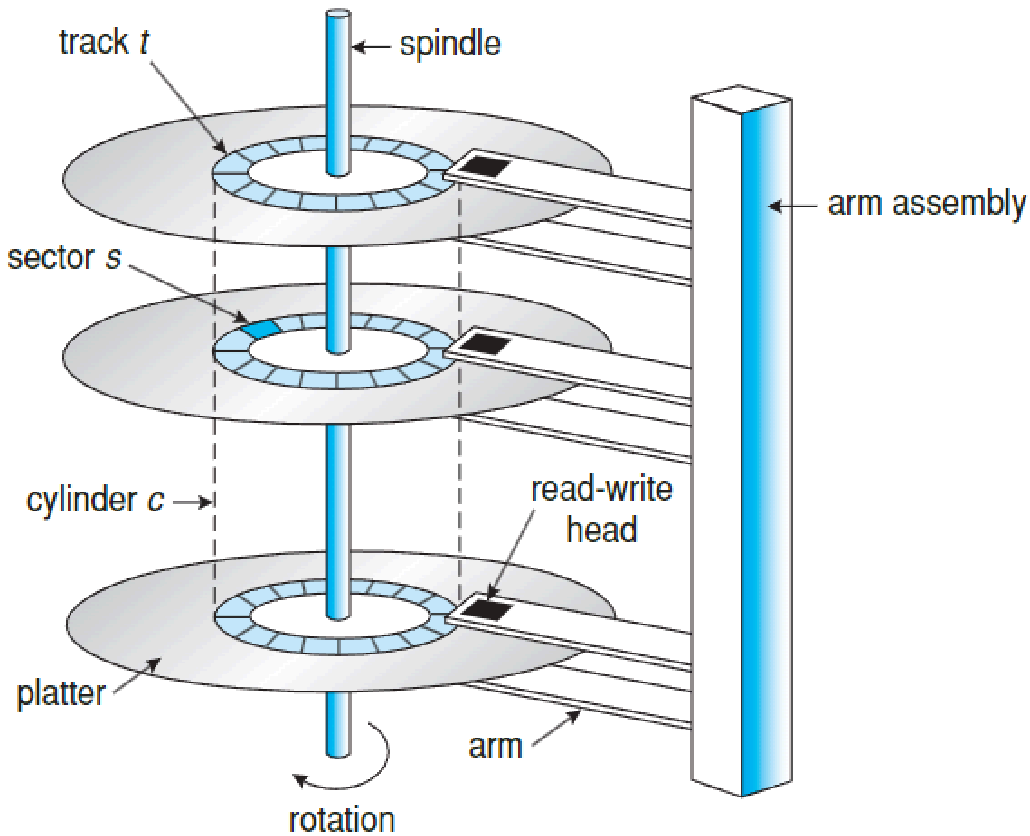
- **What it is**:
    - Memory used by the operating system for managing processes, resources, and I/O.
- **Allocation Strategies**:
    1. **Slab Allocator**:
        - Divides memory into slabs, pre-allocated for specific data structures.
    2. **Buddy System**:
        - Allocates memory in powers of 2 for flexibility.
- **Key Consideration**:
    - Kernel memory must be protected and efficiently managed to ensure system stability.

Sample Question:

- Compare the slab allocator and buddy system for managing kernel memory.

Mass Storage Structure

Hard Disk Structure



Components:

- **Platters**: Circular disks that store data.
- **Tracks**: Concentric circles on the platter surface.
- **Sectors**: Subdivisions of tracks, the smallest storage unit.
- **Cylinders**: Vertical alignment of tracks across multiple platters.
- **Read/Write Head**: Moves across tracks to read/write data.
- **Spindle**: Rotates the platters.

- **Access Time**:
  - **Seek Time**: Time to move the read/write head to the correct track.
  - **Rotational Latency**: Time waiting for the desired sector to rotate under the head.

- **Transfer Time**: Time to transfer data once the sector is under the head.

Sample Question:

- Explain the structure of a hard disk and how access time is calculated.

Disk Scheduling

Disk scheduling determines the order in which disk I/O requests are serviced to optimize performance.

Algorithms:

1. **FCFS (First-Come-First-Serve)**:
    - Services requests in the order they arrive.
    - Simple but may lead to long seek times.
2. **SSTF (Shortest Seek Time First)**:
    - Services the request closest to the current head position.
    - Reduces seek time but can lead to starvation for far-off requests.
3. **SCAN (Elevator Algorithm)**:
    - Moves the head in one direction, servicing requests along the way, then reverses direction.
    - Reduces starvation compared to SSTF.
4. **C-SCAN (Circular SCAN)**:
    - Similar to SCAN but only services requests in one direction, then jumps back to the beginning.
    - Provides more uniform wait times.
5. **LOOK and C-LOOK**:
    - Variants of SCAN and C-SCAN that only move as far as the last request in each direction, skipping unnecessary movements.

Sample Question:

- Given a disk queue with requests at tracks 98, 183, 37, 122, 14, 124, 65, and 67 (head starts at 53):
    - Calculate the total seek time for FCFS, SSTF, and SCAN.
    - Which algorithm performs best?

Calculating Average I/O Time (Assignment 13)

- **Formula**:

$$\text{Average I/O Time} = \text{Average Seek Time} + \text{Average Latency} + \left(\frac{\text{Data to Transfer}}{\text{Transfer Rate}}\right) + \text{Controller Overhead}$$

- **Components**:

    1. **Average Seek Time**: Time to position the head to the desired track.
    2. **Average Latency**: Time for the desired sector to rotate under the head.
        - For a disk rotating at R revolutions per minute (RPM): [\text{Average Latency} = \frac{1}{2} \times \frac{60}{R}]
    3. **Transfer Time**: Depends on the transfer rate and the size of the data.
        - [\text{Transfer Time} = \frac{\text{Data Size}}{\text{Transfer Rate}}]
    4. **Controller Overhead**: Additional time introduced by the disk controller.

Example Calculation:

- Disk: 7,200 RPM, 5ms seek time, 1Gbps transfer rate, 4KB block size, 0.1ms controller overhead.
    - Average Latency: [\frac{1}{2} \times \frac{60}{7,200} = 4.17\text{ms}]
    - Transfer Time: [\frac{4 \times 8 \text{Kb}}{1,000 \text{Mbps}} = 0.032\text{ms}]
    - Total I/O Time: [5 + 4.17 + 0.032 + 0.1 = 9.302\text{ms}]

Sample Question:

- Calculate the average I/O time for a disk with:
    - 15,000 RPM
    - Seek time: 3ms
    - Transfer rate: 500MBps
    - Data block size: 64KB
    - Controller overhead: 0.2ms
