Memory Management

Memory in a Computer System:

1. Basic Hardware:
    - Memory is central to the operation of a modern computer system.
    - It consists of a large array of bytes, each with its own address.
    - The typical instruction-execution cycle involves fetching an instruction from memory, decoding it, and fetching operands from memory if necessary. After execution, results may be stored back in memory.
2. Address Binding:
    - Address binding involves mapping the logical addresses used by a program to physical addresses in the computer's memory.
    - It includes compile time, load time, execution time address binding methods to associate addresses during different stages.
3. Logical Versus Physical Address Space:
    - An address generated by the CPU is referred to as a logical address or virtual address.
    - The address loaded into the memory-address register of the memory is called a physical address.
    - The set of all logical addresses generated by a program constitutes the logical address space.
    - The set of physical addresses corresponding to logical addresses forms the physical address space.

Logical versus Physical Address Space

- An address generated by the CPU is commonly referred to as a logical address or virtual address.
- An address seen by the memory unit, loaded into the memory address register, is commonly referred to as a physical address.
- The set of all logical addresses generated by a program forms a logical address space.
- The set of all physical addresses corresponding to these logical addresses is a physical address space.

Swapping

What is Swapping?

- A process in memory can be temporarily moved to a backing store and then brought back for execution.
- Enables the total physical address space to exceed real physical memory, increasing multiprogramming.

1. Standard Swapping

- Involves moving processes between main memory and a backing store (commonly a fast disk).
- Backing store must be large enough for all users' memory images.
- Ready Queue: Maintains processes with memory images on the backing store or in memory and ready to run.
- Dispatcher: Swaps processes based on CPU scheduler's decision, reloads registers, and transfers control.

Factors:

- High context-switch time.
- Total transfer time proportional to swapped memory.
- Process must be completely idle for swapping.

Standard Swapping in Modern Operating Systems

- Not used due to high swapping time and minimal execution time.
- Modified versions found in UNIX, Linux, and Windows.
- Swapping is disabled, starting only if free memory falls below a threshold, halting when free memory increases.
- Another variation swaps portions of processes to reduce swap time.
  2. Swapping on Mobile Systems

Reasons for Lack of Swapping:

- Mobile systems often use flash memory, not hard disks, limiting space and avoiding swapping.
- Flash memory has a limited write tolerance before becoming unreliable.
- Poor throughput between main memory and flash memory in mobile devices.

Mechanisms instead of Swapping:

- Apple's iOS:
  - Asks applications to voluntarily release allocated memory.

    - Failing to free up sufficient memory may result in termination by the operating system.
- **Android:**
    - Does not support swapping.
    - Adopts a strategy similar to iOS.
    - May terminate a process with insufficient free memory.
    - Writes the application state to flash memory before termination for quick restart.

Contiguous Allocation

Contiguous Memory Allocation

1. **Memory Protection:**
    - Relocation register: Contains the smallest physical address.
    - Limit register: Contains the range of logical addresses (e.g., relocation = 100040, limit = 74600).
    - Logical addresses must fall within the limit register range.
    - MMU maps logical addresses by adding the relocation register value.
    - During context switch, dispatcher loads relocation and limit registers.
    - Protects the operating system and other user programs and data from unauthorized modification.
2. **Memory Allocation Methods:**

    a. **Fixed-Sized Partitions:**

    - Divides memory into fixed-sized partitions, each containing one process.
    - When a partition is free, a process is loaded into it.
    - Partition becomes available when the process terminates.
    - Original method used by IBM OS/360 (MFT) but no longer in use.

    b. **Variable Sized-Partition:**

    - Operating system maintains a table indicating available and occupied memory parts.
    - All memory initially available for user processes, considered one large block (hole).
    - When a process arrives, system searches for a hole large enough.
    - If the hole is too large, it's split; one part is allocated, and the other returns to the set of holes.
    - When a process terminates, its memory block is returned to the set of holes.
    - Adjacent holes may merge into a larger hole.

        - System checks if newly freed memory satisfies waiting processes' demands.

Dynamic Storage Allocation Problem (Memory Allocation Techniques):

- Concerns satisfying a request of size n from a list of free holes.
- Strategies: First fit, Best fit, and Worst fit.
    - First fit: Allocate the first hole big enough; searching can start from the beginning or the previous search's end.
    - Best fit: Allocate the smallest hole big enough; search the entire list unless ordered by size.
    - Worst fit: Allocate the largest hole; search the entire list unless ordered by size.
- Comparison:
    - First fit and Best fit are better than Worst fit in decreasing time and storage utilization.
    - Neither is clearly better in terms of storage utilization, but First fit is generally faster.

3. Fragmentation
    1. Internal Fragmentation:
        - Overhead to track holes is larger than the hole itself.
        - Approach: Break physical memory into fixed-sized blocks, allocating memory based on block size.
        - Allocated memory may be slightly larger than requested, causing internal fragmentation.
        - Internal fragmentation is the unused memory within a partition.
    2. External Fragmentation:
        - Both first-fit and best-fit memory allocation strategies suffer from external fragmentation.
        - Free memory space breaks into small pieces as processes are loaded and removed.
        - External fragmentation occurs when total memory space can satisfy a request, but available spaces are not contiguous.
        - 50-percent rule: Approximately one-third of memory may be unusable due to fragmentation.

Solution to External Fragmentation:
a. Compaction:
- Goal: Shuffle memory contents to place all free memory together in one large block.
- Possible only with dynamic relocation at execution time.

- Simple compaction algorithm: Move all processes toward one end, creating one large hole.
- Can be expensive.

b. Noncontiguous Logical Address Space:
- Permits noncontiguous logical address space for processes.
- Allows a process to be allocated physical memory wherever available.
- Achieved through segmentation and paging.

Paging

Paging

1. Basic Method:
    - Frames: Physical memory is divided into fixed-sized blocks called frames.
    - Pages: Logical memory is divided into blocks of the same size called pages.
    - When a process is executed, its pages are loaded into available memory frames from their source (file system or backing store).
2. Hardware Support for Paging:
    - Every CPU-generated address is divided into two parts: a page number (p) and a page offset (d).
    - Page Table: The page number serves as an index into a page table, containing the base address of each page in physical memory. The base address, combined with the page offset, defines the physical memory address sent to the memory unit.
    - Frame Table: Managed by the operating system, it keeps track of frame allocation details—whether frames are free or allocated, the total number of frames, etc. Each entry corresponds to a physical page frame.
3. Defining Page Size:
    - Page size (and frame size) is defined by the hardware, typically a power of 2, ranging from 512 bytes to 1 GB per page, depending on the computer architecture.
    - The high-order $m-n$ bits of a logical address designate the page number, and the n low-order bits designate the page offset.
    - Logical address: [p \space (page \ number) \space|\space d \space (page \ offset)]
4. Example:
    - Logical address: $n=2$, $m=4$, page size=4 bytes, physical memory=32 bytes (8 pages).
    - Logical address 0 (page 0, offset 0) maps to physical address 20 [= (5 × 4) + 0].

- Logical address 3 (page 0, offset 3) maps to physical address 23 [= (5 × 4) + 3].
- Logical address 4 (page 1, offset 0) maps to physical address 24 [= (6 × 4) + 0].
- Logical address 13 maps to physical address 9.
  Hardware Implementation

Segmentation

1. **Basic Method:**
   - Segmentation is a memory-management scheme supporting the programmer's view of memory.
   - Logical address space consists of variable-sized segments, each with a name and length.
   - Addresses specify both the segment name and the offset within the segment (two quantities: segment name and offset).
   - Segments are numbered and referred to by a segment number, forming a logical address as a two-tuple: `<segment-number, offset>`.
   - Example Segments:
     1. Code
     2. Global variables
     3. Heap (memory allocation)
     4. Stacks for each thread
     5. Standard C library
2. **Segmentation Hardware:**
   - Physical memory is a one-dimensional sequence of bytes, but segmentation provides a two-dimensional user-defined address space.
   - Implementation uses a segment table, where each entry has a segment base and segment limit.
     - **Segment Base:** Starting physical address where the segment resides.
     - **Segment Limit:** Length of the segment.
   - Logical address (s, d) is used as an index to the segment table.
   - If the offset (d) is within the segment limit, it is added to the segment base to get the physical memory address.
   - Segment table is an array of base–limit register pairs.

Example:

- Five segments numbered 0 through 4 with respective segment table entries.

- Segment 2 (400 bytes) begins at 4300; reference to byte 53 of segment 2 maps to 4353.
- Segment 3 (unknown length) begins at 3200; reference to byte 852 maps to 4052.
- Reference to byte 1222 of segment 0 (1,000 bytes long) results in a trap to the operating system.

Segmentation with Paging

1. Basic Concept:
    - Combines segmentation and paging for enhanced memory management.
    - Main memory is divided into variable-sized segments, further divided into fixed-sized pages.
    - Each segment has a page table, resulting in multiple page tables for every program.
2. Logical Address Representation:
    - Segment Number: Points to the appropriate segment.
    - Page Number: Points to the exact page within the segment.
    - Page Offset: Used as an offset within the page frame.
3. Structure:
    - Each page table contains information about pages within a segment.
    - Segment table entries point to page table entries, and each page table entry corresponds to a page within a segment.
4. Translation of Logical to Physical Address:
    - CPU generates a logical address, divided into Segment Number and Segment Offset.
    - Segment Offset must be within the segment limit.
    - Offset further divides into Page Number and Page Offset.
    - Page Number is added to the page table base to map to the exact page in the page table.
    - Frame number with the page offset maps to the main memory for the desired word in the page of the process's segment.
5. Advantages:
    1. Reduces memory usage.
    2. Page table size is limited by the segment size.
    3. One entry in the segment table corresponds to an actual segment.
    4. Eliminates external fragmentation.
    5. Simplifies memory allocation.
6. Disadvantages:
    1. Internal fragmentation persists.

2. Higher complexity compared to paging.
3. Page tables need to be contiguously stored in memory.

Virtual Memory

Demand Paging

1. Definition:
    - Loading pages into memory only when needed during program execution.
    - Commonly used in virtual memory systems.
    - Pages are loaded on demand, reducing unnecessary memory usage.
2. Lazy Swapper:
    - Similar to a paging system with swapping.
    - Uses a lazy swapper, bringing in pages only when needed.
    - Pager is employed instead of a swapper in the context of demand paging.
3. Transfer of Paged Memory:
    - Basic concept involves transferring pages between main memory and contiguous disk space.
4. Basic Concepts:
    - Guessing Pages: Pager predicts pages needed before a process is swapped out, loading only necessary pages.
    - Valid–Invalid Bit: Hardware support to distinguish between in-memory and on-disk pages.
    - Page Fault: Accessing an invalid page causes a page fault, triggering an OS trap.
5. Handling Page Fault:
    1. Check internal table to determine the reference validity.
    2. If invalid, terminate the process; if valid but not in memory, page it in.
    3. Find a free frame, schedule disk operation to read the page, and update tables.
    4. Restart the interrupted instruction, now with the required page in memory.
6. Pure Demand Paging:
    - Start executing a process with no pages in memory.
    - Fault for each non-resident page, bringing them in as needed.
    - Executes with no more faults once all required pages are in memory.
7. Hardware Support:
    - Page Table: Marks entries as valid or invalid.
    - Secondary Memory: Holds non-resident pages, often a high-speed disk (swap device).

8. Performance:
    - Effective access time = (1 − p) × ma + p × page fault time.
    - Page-fault rate directly affects effective access time.
    - Disk I/O to swap space is faster than file system I/O.
9. Mobile Operating Systems:
    - Typically do not support swapping.
    - Demand-page from the file system and reclaim read-only pages if memory is constrained.
    - Anonymous memory pages are not reclaimed unless the application is terminated.

Example:

- Effective Access Time = (1 − p) × (200) + p (8 milliseconds).
- Directly proportional to the page-fault rate.
- Swap space is used for pages not associated with a file, known as anonymous memory.

Page Replacement

1. Demand Paging Benefits:
    - Pages brought into memory only when needed.
    - Saves I/O for loading unused pages.
    - Increases the degree of multiprogramming.
2. Over-Allocating Memory:
    - Risk of over-allocating memory with many processes, leading to potential issues.
    - Demand paging avoids loading all pages, allowing more processes despite potential memory over-allocation.
3. Over-Allocation Issues:
    - Scenario: 10 processes with 10 pages each, but only 5 pages may be used.
    - Without demand paging, only 5 processes can be accommodated due to loaded pages.
    - Demand paging allows all 10 processes, but a problem arises if a process suddenly needs all 10 pages.
4. Basic Page Replacement:
    - Approach:
        1. Find the location of the desired page on the disk.
        2. Find a free frame:
            a. Use a free frame if available.

b. If none, apply a page-replacement algorithm to select a victim frame.
    c. Write the victim frame to the disk; update tables.
3. Read the desired page into the freed frame; update tables.
4. Continue the user process from the page fault location.

5. **Modify Bit (Dirty Bit):**
    - Reduces overhead when no frames are free.
    - Modify bit indicates if a page has been modified.
    - If set, the page is written to disk; if not set, no write is needed.
    - Helps avoid unnecessary writes and reduces page-fault service time.
6. **Major Problems in Demand Paging Implementation:**
    - Frame Allocation Algorithm: Deciding how many frames to allocate to each process.
    - Page Replacement Algorithm: Selecting frames to be replaced when needed.
7. **Reference String:**
    - String of memory references used to evaluate page-replacement algorithms.
    - Can be generated artificially or traced from a system.
    - Considers only the page number, not the entire address.
    - Immediate consecutive references to the same page do not cause additional page faults.
8. **Example:**
    - Reference String: 1, 4, 1, 6, 1, 6, 1, 6, 1, 6, 1 (at 100 bytes per page).
    - Evaluation of algorithms based on the number of page faults in reference strings.

Page Replacement Algorithms

- FIFO First In First Out
- LRO Last Recently Used
- Optical Algorithm