Introduction to Computer Science – D684

Section 4

Lesson 1

* 1. **– Operating Systems**

**Modern software can be divided into 2 categories**

|  |  |  |  |
| --- | --- | --- | --- |
| Category | Definition | Functions | Examples |
| **Application** Software | Software designed for end users to perform specific tasks | -Solves real world problems  -Provides user-friendly interfaces  -Runs on top of system software | Word processors, games, inventory management, diagnostic software, missile guidance systems |
| **System** Software | Software that manages hardware and provides an environment for application software | -controls system resources  -ensures efficient execution of applications  -provides a platform for other software | Operating systems (Windows, macOS, Linux), device drivers, utility programs, firmware, ROM (Read Only Memory) |

Booting process:

1. System is turned on ->
2. one operating system becomes active and takes control ->
3. computer hardware has been wired to initially load a small set of system instructions that is stored in permanent memory (ROM) ->
4. these instructions load a larger portion of system software from secondary memory (usually a magnetic disk) ->
5. eventually all key elements of OS software are loaded ->
6. startup programs are executed ->
7. user interface is activated ->
8. system is ready for use

**Key Concepts: Memory, Process, and CPU Management**

|  |  |  |
| --- | --- | --- |
| Concept | Definition | Real-Life Example |
| **Multiprogramming** | Running multiple programs in memory simultaneously to improve CPU utilization | A user streams music while browsing the internet and running a spreadsheet program |
| **Memory Management** | Allocates memory to processes and ensures efficient use of system memory | A smartphone manages RAM by suspending background apps to keep the system responsive |
| **Process** | A program in execution, including its code, data, and system resources | Opening a web browser starts a **process**, which loads the interface, user settings, and web pages |
| **Process Management** | The OS schedules and coordinates processes to ensure smooth execution | A laptop runs multiple browser tabs, each as a separate **process**, switching between them as needed |
| **CPU Scheduling** | Determines which process gets CPU time, ensuring fair resource distribution | A video editing software runs complex tasks in the background while the user types an email |
| **Batch Processing** | Processes multiple jobs in a queue without user interaction | A bank processes thousands of transactions overnight instead of handling them individually; payroll |
| **Time-Sharing** | Multiple users share system resources by rapidly switching between tasks | An office server allows multiple employees to access shared documents and email simultaneously |

|  |  |  |  |
| --- | --- | --- | --- |
| Category | Definition | Key Characteristics | Real-Life Example |
| **Virtual Machines (VMs)** | A software-based simulation of a physical computer that runs an independent OS within another OS | - Provides isolation between the guest OS and host OS - Can run multiple OS on a single machine - Often used for software testing, development, and cloud computing | Running Windows inside macOS using software like VirtualBox or VMware |
| **Mainframes** | Large, powerful computers used for bulk data processing and supporting multiple users simultaneously | - Optimized for high-speed processing and reliability - Supports hundreds or thousands of users concurrently - Often used in enterprises and government agencies | Banking systems, airline reservation systems, and government databases |
| **Dumb Terminals** | Basic input/output devices with no independent processing power that rely on a central computer (mainframe or server) | - Only displays information and sends user inputs to the mainframe - Cannot run applications independently - Used in environments where central control is needed | Early ATMs, point-of-sale (POS) systems in retail stores, or early library catalog systems |

|  |  |  |
| --- | --- | --- |
| Term | Definition | Real-Life Example |
| **Real-Time System** | A computer system that processes data and provides output within a strict time constraint to ensure immediate responses to external events. Used in environments where delays could cause system failure or harm. | Airbag deployment in a car crash detection system, where the airbag must deploy within milliseconds to protect passengers. |
| **Response Time** | The time interval between an input or event occurring and the system's response. In real-time systems, response time must be predictable and within strict limits. | A pacemaker monitors a heartbeat and delivers an electrical pulse immediately if an irregular rhythm is detected. |

**There are 3 techniques for Memory Management:**

|  |  |  |
| --- | --- | --- |
| Memory Management Technique | Definition | Key Concepts and Applicability |
| **Single Contiguous Memory Management** | The entire program is loaded into one single block of memory, with the OS in a fixed portion and all remaining memory allocated to a single program | - **Base Register:** Stores the starting address of the program.  - **Bounds Register:** Stores the limit of the program’s memory to prevent accessing out-of-bound memory.  - **Applicability:** Applies here but is more relevant in partitioned memory |
| **Partition Memory Management** | Divides memory into sections to allow multiple programs to run at the same time. Partitions can be fixed or dynamic | - **Base Register:** Holds the starting address of each partition.  - **Bounds Register:** Ensures processes do not exceed their allocated memory.  - **Applicability:** Used in both fixed and dynamic partitions |
| **Paged Memory Management** | Divides process memory into fixed-size pages and divides physical memory into frames of the same size. A page table maps pages to frames | A program of 12KB divided into 4KB pages would need 3 pages, which may be stored in non-contiguous frames |
| Key Terms for Paged Memory Management: |  |  |
| Frames | Fixed-sized blocks of physical memory where pages are loaded | If RAM has 16 frames and each frame is 4KB, total available memory is 64KB |
| Pages | Fixed-sized blocks of a process's logical memory that match the size of frames | A 10KB process divided into 4KB pages will need 3 pages (2 full + 1 partially used) |
| Page-Map Table (PMT) | A table maintained by the OS that keeps track of which logical pages are stored in which physical frames | When a program accesses page 2, the PMT directs it to the correct frame in RAM |
| Demand Paging | Pages are loaded into memory only when needed, reducing memory usage. | If a process is 50MB but only 10MB is needed at the moment, only those pages are loaded |
| Page Swap | Moving pages between RAM and disk storage when needed | If RAM is full, a page not in use may be swapped out to the hard drive to make space |
| Virtual Memory | A technique that gives the illusion of a large memory space by using disk storage when RAM is full | A computer with 8GB of RAM may use part of the hard drive as “virtual memory” when running large applications |
| Thrashing | When excessive paging slows down the system due to constant swapping of pages between RAM and disk | If a system is running many applications and RAM is full, it may spend more time swapping than executing tasks |

**First vs Best vs Worst Fit**

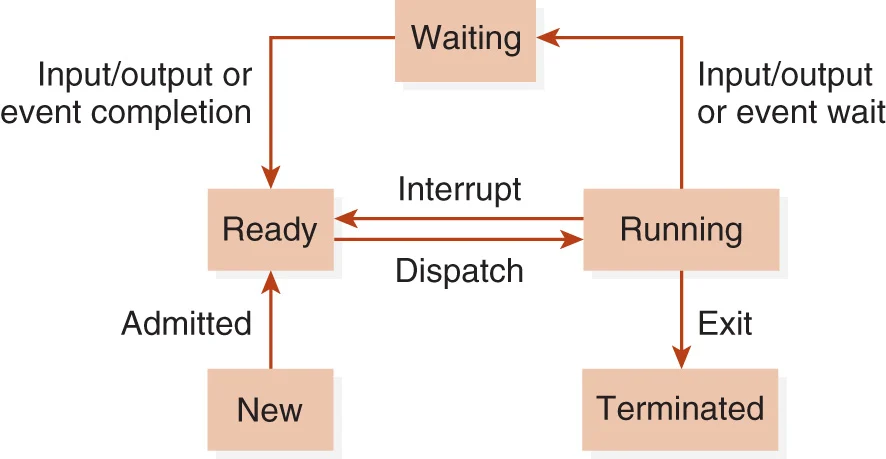
|  |  |  |
| --- | --- | --- |
| Memory Allocation Strategy | Definition | Example |
| **First Fit** | Allocates the first available memory block that is large enough for the process | If there are blocks of sizes 10MB, 4MB, and 20MB, a 3MB process will take the 4MB block |
| **Best Fit** | Allocates the smallest available block that fits the process to minimize wasted space | If there are blocks of sizes 10MB, 4MB, and 20MB, a 3MB process will take the 4MB block |
| **Worst Fit** | Allocates the largest available block, leaving the biggest leftover fragment | If there are blocks of sizes 10MB, 4MB, and 20MB, a 3MB process will take the 20MB block, leaving 17MB free |

**Fixed vs Dynamic Partitioning**

|  |  |  |
| --- | --- | --- |
| Partition Type | Definition | Pros & Cons |
| Fixed Partitioning | Memory is divided into fixed-sized partitions at system startup. Each program is assigned to a partition that fits its size | **Pros:** Simple to implement, avoids fragmentation within partitions **Cons:** Wastes memory if a program is smaller than the partition size (internal fragmentation) |
| Dynamic Partitioning | Memory partitions are created dynamically to fit program sizes | **Pros:** Avoids internal fragmentation by fitting programs exactly **Cons:** Leads to **external fragmentation** where free memory exists but is unusable because it is scattered |

**Process States and Their Definitions**

1. **New State**
   * A process is created but has not yet been admitted into the ready queue.
   * It awaits initialization by the OS.
   * Example: A user clicks on a program icon, and the OS begins loading it into memory.
2. **Ready State**
   * The process is loaded into memory and waiting for the CPU to execute it.
   * It is not currently running but is prepared to run as soon as the CPU is available.
   * Example: Multiple applications are open on a computer, and the OS schedules which process gets CPU time next.
3. **Running State**
   * The process is actively being executed by the CPU.
   * At any given time, only one process per CPU core can be in this state.
   * Example: A web browser actively loading a page is in the running state.
4. **Waiting State**
   * The process is paused and waiting for an external event (such as input/output operations) to complete.
   * Once the event occurs, it moves back to the ready state.
   * Example: A word processor waiting for user input while typing.
5. **Terminated State**
   * The process has completed execution or has been forcefully stopped.
   * It is removed from the process table and its resources are freed.
   * Example: Closing an application after use.

Figure 10.9 from page 342, Computer Science Illuminated

Note: only one process can be in the running state, many can be in the ready or waiting state at the same time

**Process Control Block (PCB)** = a data structure maintained by the OS for each process. It contains essential information about the process, allowing the OS to manage and track its execution.

**Components of a PCB:**

* **Process ID (PID):** Unique identifier for each process.
* **Process State:** Indicates whether the process is new, ready, running, waiting, or terminated.
* **Program Counter:** Keeps track of the next instruction to be executed.
* **CPU Registers:** Stores values of registers when a process is suspended.
* **Memory Management Information:** Details about allocated memory, such as page tables and base/bound registers.
* **I/O Status Information:** List of devices assigned to the process.
* **Accounting Information:** Tracks CPU usage, execution time, and other performance metrics.

**Context switching** = the process of saving the state of a currently running process and restoring the state of another process so execution can resume where it left off. The **PCB plays a critical role** in this by storing and restoring process information during switches.

**Steps of Context Switching:**

1. The OS saves the current process’s state (registers, program counter, etc.) into its **PCB**.
2. The OS selects another process from the ready queue.
3. The OS loads the new process’s saved state from its **PCB** into the CPU.
4. The new process resumes execution.

**Example:**

* If a user switches from typing in a document (Process A) to a web browser (Process B), the OS saves Process A’s state into its PCB and loads Process B’s PCB into the CPU.

**Preemptive vs Nonpreemtive Scheduling**

|  |  |  |  |
| --- | --- | --- | --- |
| Scheduling Type | Definition | Key Characteristics | Example |
| **Nonpreemptive Scheduling** | The CPU executes a process until completion or voluntary release. The OS does not forcibly remove a process from execution. | - Once a process starts, it runs until it finishes or moves to a waiting state. - Simpler but can lead to inefficiencies if long processes block shorter ones. | A print job in a queue: The printer completes the first job before moving to the next. |
| **Preemptive Scheduling** | The CPU can forcibly remove a process to allow another higher-priority or time-sliced process to execute. | - The OS can interrupt and switch between processes. - Allows better responsiveness for high-priority or short processes. - Introduces context-switching overhead. | A computer playing music while opening multiple applications—music playback (lower priority) might be paused when a CPU-intensive task starts |

**CPU Scheduling Algorithms**

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Definition | Pre or Nonpre? | Example |
| **First Come, First Served (FCFS)** | The first process in the queue gets the CPU and runs until completion before the next process starts | Nonpreemptive | Checkout line at a store: The first customer is served before the next in line |
| **Shortest Job Next (SJN) / Shortest Job First (SJF)** | The process with the shortest expected execution time is chosen first | Nonpreemptive (but can be preemptive in some cases) | A restaraunt serving small orders first to clear the queue quickly |
| **Round Robin (RR)** | Each process gets a fixed time slice (quantum). If it doesn’t finish in that time, it moves to the back of the queue | Preemptive | A multiplayer online game server allocating a few milliseconds of processing to each player before switching |

Lesson 2

**2.1– File Systems**

Memory

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Definition | Characteristics | Examples |
| **Main Memory (Primary Storage)** | Holds active programs and data while in use | - **Volatile**: Data is lost if power is turned off - Provides fast access for processing | - RAM (Random Access Memory) - Cache memory |
| **Secondary Storage** | Provides permanent, nonvolatile storage for data and programs | - **Nonvolatile**: Retains data even when power is off. - Typically larger but slower than main memory | - Hard disk drives (HDDs) - Solid-state drives (SSDs) - Portable disks (USB drives, external hard drives) - Tape drives (used mainly for archival storage) |

**File System Concepts**

* **Directory:**
  + A structure that organizes and stores files within a storage system.
  + Can contain files and other directories (subdirectories).
  + Helps manage access and retrieval of files efficiently.
* **File System:**
  + Manages how data is stored and retrieved from a storage device.
  + Handles file naming, structure, and access permissions.
  + Examples of file systems: NTFS (Windows), FAT32, ext4 (Linux), APFS (Mac).
* **Files:**
  + A collection of related data stored on a computer.
  + Operations: create, delete, open, close, read data, write data, rename, copy, truncate, reposition the current file pointer in a file, append data to the end of a file
  + **Two main types:**
    - **Text files:** Contain human-readable characters (e.g., .txt, .csv, .html).
    - **Binary files:** Contain machine-readable data (e.g., .exe, .jpg, .mp3).

**File Access Methods**

|  |  |  |
| --- | --- | --- |
| Access Method | Definition | Use Case |
| **Sequential Access** | Data is read in order, from the beginning to the end | Tape storage, logs, audio/video streaming |
| **Direct (Random) Access** | Data can be read or written at any location without following a sequence | Hard drives, databases, file editing applications |

**Directory Types and Paths**

|  |  |  |
| --- | --- | --- |
| Term | Definition | Example/Notes |
| **Directory Tree** | A hierarchical structure used to organize files and directories | - Resembles an inverted tree where the **root directory** is at the top - Branches represent subdirectories and files |
| **Root Directory** | The top-level directory in a directory tree. All other directories stem from it | - Represented as / in Unix/Linux or C:\ in Windows |
| **Working Directory** | The current directory where the user is located in the file system | - The pwd (print working directory) command in Unix/Linux shows the current directory |
| **Absolute Path** | The full path from the root directory to a specific file or directory | - **Windows:** C:\Users\John\Documents\file.txt - **Linux/macOS:** /home/john/Documents/file.txt |
| **Relative Path** | Specifies the location of a file or directory in relation to the current working directory | - **Example (if working directory is /home/john):** Documents/file.txt (instead of /home/john/Documents/file.txt) |

**Disk Scheduling Overview**

**Disk scheduling** = the process of determining the order in which disk I/O requests are handled to optimize performance, minimize seek time, and improve system efficiency

- disk access speed depends largely on the movement of the disk's read/write head,

-different algorithms are used to reduce unnecessary movement and improve response time

**Single Disk (Platter-Based Structure)**

* A **single disk** consists of circular platters that store data magnetically.
* Each platter is divided into **tracks** (concentric circles), which are further divided into **sectors** (small storage units).
* The **read/write head** moves across tracks to access data.

**Hard Disk Drive (HDD) Structure**

* A **hard disk drive (HDD)** consists of multiple stacked platters spinning around a spindle.
* Each platter has its own read/write head attached to an actuator arm, allowing multiple heads to read/write simultaneously.
* Data is stored in a hierarchical format using **cylinders, tracks, and sectors** for efficient access.

**Disk Scheduling Algorithms**

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Description | Advantages | Disadvantages |
| **First-Come, First-Served (FCFS)** | Requests are processed in the order they arrive | - Simple to implement - Fair to all requests | - Can cause long seek times if requests are scattered across the disk - Poor performance for high workloads |
| **Shortest Seek Time First (SSTF)** | The request closest to the current head position is served next | - Reduces seek time compared to FCFS - Improves efficiency | - May lead to **starvation** (far-off requests may wait indefinitely) |
| **SCAN (Elevator Algorithm)** | The disk head moves in one direction, fulfilling requests, then reverses direction when reaching the end | - Avoids starvation by servicing all requests in both directions. - Provides a more balanced approach than SSTF. | - Higher variance in response time - Can be inefficient if requests are clustered near the starting position |