5. System Software

5.1 Operating Systems

- What is an OS
  - Controls operation of computer system
  - Provides a user interface
  - Controls how computer responds to user's requests
  - Controls how hardware communicate
  - Provides interface between user and hardware
- Why a computer system requires an OS
  1. The hardware is unusable without an OS
     - hides the complexity of hardware from the user
  2. Acts as an **interface**
     - controls communications between user and hardware
  3. Provides **software platform/environment** on which programs can be run

OS Management Tasks

- **Memory Management**
  - **memory protection** to ensure two programs do not try to use the same memory space
  - use of **virtual memory**
  - deciding which processes need to be in main memory at any one time
  - **location of processes** within the memory
  - *Example*: when a process terminates, memory is made available by the OS
- **File Management**
  - maintains **directory structures**
  - provides **file naming conventions**
  - controls **access**
- **Security Management**
  - makes **provision for recovery** when data is lost
  - provides **usernames and passwords/encryption/user accounts**
  - prevents **unauthorized access**
  - ensures the **privacy of data**
- **Hardware Management (Input/Output/Peripherals)**
  - installation of **appropriate driver software**
  - controls **access to data** being sent to/from hardware/peripherals
  - controls **access to hardware/peripherals**
  - manages **communication** between devices/hardware and software
- **Process Management**
  - scheduling of **processes/multi-tasking/multi-programming**
  - resolution of **conflicts** when two or more processes require the same resource

Utility Software

- An **additional program** that helps to maintain or configure the system
- **Disk Formatter**
  - Makes existing data inaccessible

    - Partitions the disk into logical drives
    - Sets up the (specified) file system
    - Prepares the disk for initial use
    - May check for errors on the disk
- **Virus Checker**
    - Scans files stored on computer system for malicious code
    - Scans files when they enter the system or downloaded
    - Sets up schedule for virus-checking
    - Isolates or deletes viruses
    - Regularly updates virus definitions
- **Defragmentation Software**
    - Re-organizes the disk contents
    - Moves split files so they are contiguous
    - Creates a larger area of (contiguous) free space
- **Disk Contents Analysis Software**: Provides a detailed **visualization or report of the disk usage** by various files and folders. This helps in identifying large, duplicate, or unnecessary files and efficiently managing disk storage space
- **Disk Repair Software**
    - Check for any errors on the disk
    - Resolves any errors on the disk
    - Retrieves files/data from damaged disk
    - Marks bad sectors on the disk
- **File Compression**
    - Reduces file size by removing redundant data in files
    - Causes improvements in the computer's performance by reducing the data that needs to be stored
- **Back-up Software**
    - Creates copy of contents of a disk, can be set up for automatically backup
    - Allows user decide what is backed up
    - Allows user set up off site backup
    - May encrypt backup files
    - Restores data if necessary

Program Libraries

- Pre-written code that can be linked to software under development without any amendments
- Can perform common or complex tasks
- Provides ready-built routine that can be imported into a program

Benefits of Library Files

- the code is **already written**, so the programmer is not starting over again, which saves time
- the code will have been **used by many people**, so it should be already **thoroughly tested** and relatively **error-free**
- The programmer can use mathematical/graphics functions that he **may not know how to code**
- If there is an **improvement in the library routine**, the program updates automatically

Dynamic Link Library (DLL) files

- A collection of **self-contained** programs that are **already compiled**

- Linked to the main program **during execution**
- DLL code is **separate** from the executable file
- DLL files are only loaded into memory **when required at run time**
- A DLL file can be made available to **several applications** at **the same time**
- **Benefits**
    1. The executable file becomes smaller as it does not contain all the library routines, since DLL files are only loaded into memory when needed
    2. Changes/improvements/error detection to the DLL file code are done independently of the main program
        - So there is no need to recompile the main program
        - All programs using it will benefit
    3. A single DLL file can be made available to several application programs
        - Saving the space in memory
- **Drawbacks**
    1. Appropriate **linking software** must be available at run time to link/include/import the DLL files
    2. The DLL file must be present in the system, otherwise errors (unable to find X.dll files)
    3. Unexpected changes to the DL file could mean the program stops working as expected

5.2 Language Translators

Compilers, Interpreters, and Assemblers

Aspect	Assembler Software	Compiler	Interpreter
Purpose	Translates assembly language programs into machine code	Translates high-level language programs into machine code or intermediate code	Translates and executes high-level language programs line by line or statement by statement
Translation **Process**	Converts mnemonic opcode and operands into their numerical equivalents	Analyzes the entire program and converts it into a standalone executable or intermediate form	Reads, translates, and executes each line of code in sequence without producing a separate executable
Output	Machine code specific to a processor architecture	Machine code or byte code that can be executed by the machine or a virtual machine	No separate output file; the code is executed directly from the source
Speed of Execution	Generates machine code that can be directly executed by the CPU, making it very fast	Slow translation phase, but results in fast execution since the entire program is translated before execution	Slower execution compared to compiled programs, as translation occurs at runtime
Efficiency	Highly efficient as it produces code that runs directly on the CPU	Efficient in terms of execution speed after the program has been compiled	Less efficient as each instruction must be translated every time it is executed
Error Detection	Errors are detected during the assembly time, but debugging can be more difficult due to low-level code	Syntax and some semantic errors are detected at compile time; runtime errors are detected during execution	Errors are detected at runtime, line by line, which can be easier for debugging
Use Cases	Used for programs that require direct hardware manipulation, performance optimization, and small-size	Used for large applications where performance is critical and when distribution of executable is required	Used for scripting, rapid prototyping, and in situations where the environment is controlled or the program will not be widely distributed

Benefits of interpreters

- Errors can be corrected as they occur (since error occurs line by line) → Easier to debug
- Can run a partially complete program when developing
- The effect of any change made to the code can be seen immediately

Benefits of compilers

- Produces an executable file
- Users do not have access to the source code
- It will (probably) be faster to run the executable
- Code does not have to be compiled each time it is run
- Does not need the compiler to be present at run-time (so it is environment-independent)

Partial Compilation of Java

- Uses 2 step translation process
- Java will be partially compiled and then partially interpreted
- Java code is first translated to byte-code by Java compiler
- Byte-code is finally interpreted by the Java Virtual Machine to produce machine code

Integrated Development Environment (IDE)

- a single **software program** for **program editing, translation, and testing**

Features of IDE

- for **Coding**: context-sensitive prompts, auto-complete, auto-indentation
- for **Initial Error Detection**: dynamic syntax checks, highlights any undeclared variables, highlights any unassigned variables
- for **Presentation**: pretty-print (color-coding of keywords/comments, indentation), expand and collapse code blocks
- for **Debugging**: single stepping, breakpoints, report window to watch the changing value of variables