4 Software

4.1 Types of Software & Interrupts

System Software & Application Software

System Software	Application Software
Provides the services that the computer requires	Provides the services that the user requires
e.g. operating system, utility software	e.g. internet browser, games software, word processor

Utility Software (System Software)

Utility software is software designed to help **maintain, enhance** and **troubleshoot/repair a** computer system. It is designed to perform limited tasks.

- **Security utilities** (anti-virus, encryption, firewall)
- **Disk organisation utilities**
  - **System Clean-Up Tools** – remove files no longer needed, reduce space and speed up system
  - **Disk Defragmentation Tools** – rearrange the parts of files on the disk drive
    - When a file is saved to the **HDD**, parts of the file might be saved in different areas of the disk → these tools try to move all the parts to the same area for quicker access
- **Data compression utilities** (makes files smaller to take up less storage space for faster transmission)
- **File backup utilities** (backs up files and software on the system)

Operating Systems (System Software)

An operating system (**OS**) is software that **manages computer hardware** and **provides a platform for running applications.**

It provides **an interface between the user and the hardware** in a computer system

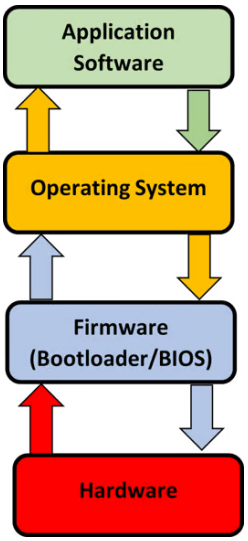
Functions of Operating Systems (OS)

- **Manage files** – use file naming conventions, control permissions, maintain directory structures
  - File management gives the user the ability to:
    - Create files, Name files, Rename files, Copy files, Delete files
- **Handle interrupts**
- **Provide a HCI (human-computer interface)** – allow user to control the OS easily
- **Manage memory** – allocate **RAM** to programs, move data between RAM and HDD/SSD
- **Mange multitasking** – allow computers to carry out more than one task at a time by using time slicing

- **Provide a platform for running applications** – application programs and hardware can communicate with each other through an application programs interface (API); allocate RAM to applications
- **Provide system security** – create/delete users for the system, access level rights, security updates
- **Manage user accounts** – each user has an account with different levels of access
- **Manage hardware peripherals (input & output devices) and drivers -** managing the way peripherals (hardware) interact with software

Running of Applications

- **Application software** (provides services the user requires) are run on the **operating system**
- The **operating system** (manages main functions of the computer) is run on the **firmware**
- The **firmware** (type of software stored in ROM which includes **basic input/output system BIOS** & **bootloader**) is run on the **hardware** (physical components of a computer system)



Interrupts

An **interrupt** is a signal sent from a **device/software** to the **CPU**, to temporarily stop the **FDE cycle** so that it can service the interrupt

Types of Interrupts

- **Hardware Interrupt** - caused by a hardware device
  - e.g. pressing a key on the keyboard, moving the mouse
- **Software Interrupt** - occurs when an application stops or requests services from the OS
  - e.g. division by zero, two processes trying to access the same memory location

Operation of Interrupts

- Hardware/Software generates the interrupt
- Interrupt is given a **priority**
- Interrupt is sent to **CPU** and placed in a **queue**
- CPU stops current task to service the interrupt using an **interrupt service routine**
- If interrupt is highest priority the interrupt is processed
- Once the interrupt has been serviced, the **previous instructions** are retrieved and the process continues

What happens as a result of Interrupts:

- The system becomes more **responsive** to real-time events (like user inputs or hardware failures).
- It allows **multitasking**, by switching between different tasks quickly.
- Critical tasks are handled **immediately**, improving performance and safety in systems.

4.2 Programming Languages, Translators and IDEs

High-level & Low-level Programming Languages

High-level Language (HLL)	Low-level Language (LLL)
Languages which use English-like statements Eg: python, Java, C++	Languages that are close to the language processed by computers eg. machine code, assembly language • **Machine code** is the binary code. Instructions are directly executable by the processor. • **Assembly language** is a form of **low-level language** that uses **mnemonics** • An **assembler** is needed to translate an **assembly language** program into **machine code**
Advantages of High-level language + Easier to read and write → less prone to errors + Easier to debug & maintain → find errors in less time + Not machine-specific (portable) → can be used on many computers/operating systems	**Advantages of Low-level language** + Can directly manipulate hardware + Can use specialised hardware + No requirement for a compiler/interpreter → faster execution + More memory efficient
Disadvantages of High-level language - Cannot directly manipulate hardware - Needs to translate machine code before running - May be less efficient	**Disadvantages of Low-level language** - Hard to read and write → more prone to errors - Harder to debug - Machine dependent

Translators

A **translator** (compiler/interpreter) translates **program source code** from an **HLL** into **machine code** (binary) to be understood by the computer

Interpreter	Compiler
• Translates & executes the code line by line • Stops execution when an error is found • When corrected, carries on running from where the error occurred • A program needs to be interpreted again each time it is run • Interpreters are generally used when a program is being written in the development stage	• Translates the entire source code at once before executing it, into an executable file • Produces **error report** for the whole code – displays all errors that require correction before execution • Once a program is compiled, the machine code can be used again to perform the same task without re-compilation • Mostly used to translate the final program
Advantages of Interpreter + Requires less RAM to process the code + Easier and quicker to test/debug programs + Program will always run (only stops when error is found)	**Advantages of Compiler** + Programs take less time to execute + Compiled programs can be executed without the compiler + A compiled program can be stored ready for use
Disadvantages of Interpreter - Programs can take longer to execute - Programs cannot be run without the interpreter - Every time the program is run it has to be translated	**Disadvantages of Compiler** - Changes mean it must be recompiled - Harder to debug (does not identify where error is) - There must be enough memory space for code to compile

Integrated Development Environment (IDEs)

An **IDE** is software used by programmers to aid the writing and development of programs

Functions:

- **Code editing** allows users to write and manipulate source code without a separate text editor (increases efficiency)
- **Run-time environment** allows the program to run and see its corresponding output
- **Translators** are provided to compiler or interpret the code and enables the program to be executed
- **Debugger** allows the programmer to step through the program line by line and identify errors
- **Error diagnostics** identify errors as the program is being typed and provide error messages
- **Auto-correction** provides user with suggested corrections for errors
- **Auto-completion** for variable names and reserved words
- **Prettyprinting** colour codes the words in the program and lays out the program in a meaningful way