Unit 8 System Software

Operating system (OS) - a set of programs that manage computer hardware resources and provide common services for application software.

The operating system acts as an interface between the hardware and the programs requesting I/O. It is the most fundamental of all system software programs.

Responsibilities of the OS include:

- Hiding the complexities of hardware from the user.
- Managing between the hardware's resources which include the processors, memory, data storage and I/O devices.
- Handling "interrupts" generated by the I/O controllers.
- Sharing of I/O between many programs using the CPU.

The most well known Operating Systems include:

- **System Software** - programs that manage the operation of a computer.
- **Application Software** - programs that help the user perform a particular task.

Different operating systems have different approaches to file management. You can also install additional file management software on most computers. But deep down, the operating system has to keep track of where files are so that software will run on your computer. That way, when a program asks for a file, the operating system knows exactly where to go to get the information.

User Interface

The User Interface is the interaction between the User and the Machine, letting the user send commands with the expected results. ***Two forms of the Interface User are the Command Line Interface (CLI) and the Graphical User Interface (GUI).***

Tasks of Operating System

The operating system's tasks, in the most general sense, fall into following categories:

1- Program-hardware Interface

The operating system has to ensure that the hardware does what the software wants it to do. Program development tools associated allow a programmer to write a program without needing to know the details of how the hardware, particularly the processor, actually works. The operating system then has to provide the mechanism for the execution of the developed program.

2- Processor Management

The heart of managing the processor comes down to two related issues:

- Ensuring that each process and application receives enough of the processor's time to function properly
- Using as many processor cycles as possible for real work

The basic unit of software that the operating system deals with in scheduling the work done by the processor is either a **process** or a **thread**, depending on the operating system.

Process: a program that has begun execution.

Thread: smallest independent sequence of a program instruction.

3- Memory Storage and Management

When an operating system manages the computer's memory, there are two broad tasks to be accomplished:

1. Each process must have enough memory in which to execute, and it can neither run into the memory space of another process nor be run into by another process.
2. The different types of memory in the system must be used properly so that each process can run most effectively.

4- Device Management

1. Installation of the appropriate device driver software
2. Control of usage by processes.

5- File Management

1. file naming conventions
2. Directory (folder) structures
3. Access control mechanisms.

6- Security Management

1. Provision for recovery when data is lost
2. Prevention of intrusion
3. Ensuring data privacy.

7- Error Detection and Recovery

Errors can arise in the execution of a program either because it was badly written or because it has been supplied with inappropriate data. Other errors are associated with devices not working correctly. Whatever the cause of an error, the operating system should have the capability to interrupt a running process and provide error diagnostics where appropriate. In extreme cases, the operating system needs to be able to shut down the system in an organized fashion without loss of data.

Utility Programs

Utility software is a type of system software which has a very specific task to perform related to the working of the computer, for example anti-virus software, disk defragment etc.

Utility software should not be confused with application software, which allows users to do things like creating text documents, playing games, listening to music or surfing the web. Rather than providing these kinds of user-oriented or output-oriented functionality, utility software usually focuses on how the computer infrastructure (including the computer hardware, operating system, application software and data storage) operates. Due to this focus, utilities are often rather technical and targeted at people with an advanced level of computer knowledge.

Examples of utility software include:

1. Virus scanner :

It provides permanent facility to protect a computer system. When a new virus comes along there is a delay before it is recognized and a further delay before a virus checker has been updated to deal with it. As a result, it is necessary for a virus checker to be regularly updated and for it to scan all files on a computer system as a matter of routine.

2. Disk defragmenter - to speed up your hard disk.

Disks gradually become less efficient because the constant creation, editing and deletion of files leaves them in a fragmented state.

3. Hard Disk Formatter and Checker

A disk formatter will typically carry out the following tasks:

1- Removing existing data from a disk that has been used previously.
2- Setting up the file system on the disk, based on a table of contents that allows a file recognized by the operating system to be associated with a specific physical part of the disk
3- Partitioning the disk into logical drives if this is required.

4. File managers - to add, delete, rename and move files and folders.

5. Backup software

It is quite likely that you perform a manual backup of your own files every now and then using a flash memory stick. However, an easier way to perform backup is to use a backup utility program. Such a program will:

1- Establish a schedule for backups

2- Only create a new backup file when there has been a change.

6. File Compression

Library programs

Library programs are collections compiled routines which are shared by multiple programs, such as the printing function.

Library programs contain code and data that provide services to other programs such as interface (look and feel), printing, network code and even the graphic engines of computer games. If you have ever wondered why all Microsoft Office programs have the same look and feel, that is because they are using the same graphical user interface libraries. For computer games a developer might not have the time and budget to write a new graphics engine so they often buy graphical libraries to speed up development, this will allow them to quickly develop a good looking game that runs on the desired hardware.

Most programming languages have a standard set of libraries that can be used, offering code to handle input/output, graphics and specialist math functions. You can also create your own custom libraries and when you start to write lots of programs with similar functionality, you'll find them very useful.

Translator software

Assembler

An **assembler** translates assembly language into machine code. Assembly language consists of mnemonics for machine opcodes so assemblers perform a 1:1 translation from mnemonic to a direct instruction. For example:

`LDA #4` converts to `0001001000100100`

Conversely, one instruction in a high level language will translate to one or more instructions at machine level.

Advantages of using an Assembler:

- Very fast in translating assembly language to machine code as 1 to 1 relationship
- Assembly code is often very efficient (and therefore fast) because it is a low level language
- Assembly code is fairly easy to understand due to the use of English-like mnemonics

Disadvantages of using Assembler:

- Assembly language is written for a certain instruction set and/or processor
- Assembly tends to be optimized for the hardware it's designed for, meaning it is often incompatible with different hardware
- Lots of assembly code is needed to do relatively simple tasks, and complex programs require lots of programming time

Compiler

A **Compiler** is a computer program that **translates code** written in a high level language to a lower level language, object/machine code. The most common reason for translating source code is to create an executable program (converting from a high level language into machine language).

Advantages of using a compiler

- Source code is not included, therefore compiled code is more secure than interpreted code
- Tends to produce faster code than interpreting source code
- Produces an executable file, and therefore the program can be run without need of the source code

Disadvantages of using a compiler

- Object code needs to be produced before a final executable file, this can be a slow process
- The source code must be 100% correct for the executable file to be produced.

For a compiler the following steps apply:

- **1** The compiler program and the source code file are made available but no data is needed.
- **2** The compiler program begins execution.
- **3** The first line of the source code is read.
- **4** The line is analyzed.
- **5** If an error is found this is recorded.
- **6** If no error is found the line of source code is converted to an intermediate code.
- **7** The next line of source code is read and Steps 4-7 are repeated.
- **8** when the whole of the source code has been dealt with one of the following happens:
- If no error is found in the whole source code the complete intermediate code is converted into object code.
  If any errors are found a list of these is output and no object code is produced.

- Execution of the program can only begin when the compilation has shown no errors. This can take place automatically under the control of the compiler program if data for the program is available.

Interpreter

An interpreter program executes other programs directly, running through program code and executing it line-by-line. As it analyses every line, an interpreter is slower than running compiled code but it can take less time to interpret program code than to compile and then run it — this is very useful when prototyping and testing code. Interpreters are written for multiple platforms, this means code written once can be run immediately on different systems without having to recompile for each. Examples of this include flash based web programs that will run on your PC, MAC, games console and Mobile phone.

Advantages of using an Interpreter

- Easier to debug(check errors) than a compiler
- Easier to create multi-platform code, as each different platform would have an interpreter to run the same code
- Useful for prototyping software and testing basic program logic

Disadvantages of using an Interpreter

- Source code is required for the program to be executed, and this source code can be read making it insecure
- Interpreters are generally slower than compiled programs due to the per-line translation method

For an interpreter the following steps apply:

- **1** The interpreter program, the source code file and the data to be used by the source code program are all made available.
- **2** The interpreter program begins execution.
- **3** The first line of the source code is read.
- **4** The line is analyzed.
- **5** If an error is found this is reported and the interpreter program halts execution.
- **6** If no error is found the line of source code is converted to an intermediate code.
- **7** The interpreter program uses this intermediate code to execute the required action.
- **8** The next line of source code is read and Steps 4-8 are repeated.

Commercial software

Commercial software almost always has to be paid for but there are a number of different options that might be available:

1. A fee is paid for each individual copy of the software.
2. A company might have the option of buying a site license which allows a defined number of copies to be running at any one time.
3. Special rates might be available for educational use.
4. Earlier versions or limited versions might be offered free or at reduced price.

Open source software, including the source code, available for free. The user of the software is free to use it, modify it, copy it or distribute it according to need.

Freeware This is software that is distributed for free but without the source code.

Shareware: software free for use for a limited period but no source code provided.

Java

When the programming language Java was created, a different philosophy was applied to how it should be used. Each different type of computer has to have a Java Virtual Machine created for it. Then when a programmer writes a Java program this is compiled first of all to create what is called Java Byte Code. When the program is run, this code is interpreted by the Java Virtual Machine. The Java Byte Code can be transferred to any computer that has a Java Virtual Machine installed.

Features found in a typical Integrated Development Environment (IDE)

1- **Prettyprinting**
Prettyprint refers to the presentation of the program code typed into an editor. For example, the VB IDLE automatically color-codes keywords, built-in function calls, comments, strings and the identifier in a function header. In addition, indentation is automatic.

2- **Context-sensitive prompts**
This feature displays hints (or a choice of keywords) and available identifiers that might be appropriate at the current insertion point of the program code.

3- **Dynamic syntax checks**
When a line has been typed, some editors perform syntax checks and alert the programmer to errors.

4- **Expanding and collapsing code blocks**
When working on program code consisting of many lines of code, it saves excessive scrolling if you can collapse blocks of statements.

5- **Debugging**
An IDE often contains features to help with debugging (finding and correcting errors, often called ‘bugs’, in a program).