Chapter 8:
System software

Learning objectives

By the end of this chapter you should be able to:

- explain why a computer system requires an Operating System (OS)
- explain the key management tasks carried out by the Operating System
- show understanding of the need for typical utility software provided with an Operating System
- show understanding of program libraries
- show understanding of the need for:
  - assembler software for the translation of an assembly language program
  - a compiler for the translation of a high-level language program
  - an interpreter for translation and execution of a high-level language program
- explain the benefits and drawbacks of using either a compiler or an interpreter and justify the use of each
- show awareness that high-level language programs may be partially compiled and partially interpreted, such as Java
- describe features found in a typical Integrated Development Environment (IDE).

8.01 System software

In the 1960s, the likely arrangement for using a computer would be something like this.

1. Enter machine room with deck of punched cards and a punched paper tape reel.
2. Switch on computer.
3. Put deck of cards into card reader and press button.
4. Put paper tape into tape reader and press button.
5. Press button to run the program entered into memory from the punched cards using the data entered into memory from the paper tape.
6. Press button to get output printed on the line-printer.
7. Switch off computer.
8. Leave machine room with deck of cards, paper tape and line-printer output.

The user controlled the computer hardware by pressing buttons. Just try to imagine how many buttons would be needed if you had to control a computer in the same way today.

The missing component from the 1960s computer was, of course, an **operating system**; in other words, some software to control the hardware and interact with application software. An operating system is an example of a type of software called ‘system software’. This distinguishes it from application software, which is created to perform a specific task for a computer user rather than just helping to run the system.

8.02 Operating system activities

Operating systems are extremely complex and it is not possible to give a full description here of what an operating system is. However, what an operating system generally does is to provide an environment where programs can be run that are of benefit to a user.

The activities of an operating system can be sub-divided into different categories, some of which overlap with each other. We are going to look at each of the various tasks carried out by the operating system. Details of how some of them are carried out are discussed in Chapter 20 (Sections 20.01 to 20.05).

User–system interface

A user interface is needed to allow the user to get the software and hardware to do something useful. An operating system should provide at least the following for user input and output:

- a command-line interface
- a graphical user interface (GUI).

Discussion Point:

Have you any experience of using a command-line interface?

Program–hardware interface

Programmers write software and users run this software. The software uses the hardware. The operating system has to ensure that the hardware does what the software wants it to do. Program development tools associated with a programming language 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 running the developed program.

Resource management

When a program has started to run it is described as a **process**. In a modern computer system, a process will not be able to run to completion without interruption. At any time there will be many processes running on the computer system. Each process needs access to the resources provided by the computer system.

The resource management provided by the operating system aims to achieve the best possible efficiency in computer system use. The two most important aspects of this are:

- scheduling of processes
- resolution of conflicts when two processes require the same resource.

Memory management

There are three important aspects of memory management.

- Memory protection ensures that one program does not try to use the same memory locations as another program.
- The memory organisation scheme is chosen to achieve the best usage of limited memory size, for example, virtual memory involving paging or segmentation.
- Memory usage optimisation involves decisions about which processes should be in main memory at any one time and where they are stored in this memory.

Device management

Every computer system has a variety of components that are categorised as ‘devices’. Examples include the monitor screen, the keyboard, the printer and the webcam. The management of these requires:

- installation of the appropriate device driver software
- control of usage by processes.

File management

Three major features here are the provision of:

- file naming conventions
- directory (folder) structures
- access control mechanisms.

Security management

Chapter 9 (Sections 9.02, 9.03 & 9.04) and Chapter 21 contain extensive accounts of security issues and measures used to prevent problems. For the operating system the many aspects of security management include:

- provision for recovery when data is lost
- prevention of intrusion
- ensuring data privacy.

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 organised fashion without loss of data.

TASK 8.01

For each of the above categories of operating system task, each point could be placed in a different category. Make an abbreviated list of these categories and add arrows to show different categories where each point could be placed.

Question 8.01

It is useful to describe the management tasks carried out by an operating system as being primarily one of the following types:

- those assisting the user of the system
- those concerned with the running of the system.

Considering the management tasks that have already been categorised, can you identify them as belonging to one or other of the above types? Are there any problems in doing this?

8.03 Utility software

A utility program can be provided by the operating system or it can be installed separately. It is a program that is not executed as part of the normal running of the operating system. Instead it is a program that the user or the operating system can decide to run when needed. For example, some utility programs manage hard disks.

Hard disk formatter and checker

A disk formatter will typically carry out the following tasks:

- removing existing data from a disk that has been used previously
- setting up the file system on the disk, based on a table of contents that allows a file recognised by the operating system to be associated with a specific physical part of the disk
- partitioning the disk into logical drives if this is required.

Another utility program, which can be a component of a disk formatter, performs disk contents analysis and, if possible, disk repair when needed. The program first checks for errors on the disk. Some errors arise from a physical defect resulting in what is called a 'bad sector'. There are a number of possible causes of bad sectors. However, they usually arise either during manufacture or from mishandling of the system. An example is moving the computer without ensuring that the disk heads are secured away from the disk surface.

Other errors arise from an event such as a loss of power or an error causing sudden system shutdown. As a result some of the files stored on the disk might no longer be useable. A disk repair utility program can mark bad sectors and ensure that the file system no longer tries to use them. When the integrity of files has been affected, the utility might be able to recover some of the data but otherwise it has to delete the files.

Hard disk defragmenter

A disk defragmenter utility also can be part of a disk repair utility program but it is not primarily concerned with errors. A perfectly functioning disk will, while in use, gradually become less efficient because the constant creation, editing and deletion of files leaves them in a fragmented state. The cause of this is the logical arrangement of data in sectors as discussed in Chapter 3 (Section 3.04), which does not allow a file to be stored as a single block of data.

A simple illustration of the problem is shown in Figure 8.01. Initially file A occupies three sectors fully and part of a fourth one. File B is small so occupies only part of a sector. File C occupies two sectors fully and part of a third. When File B is deleted, the sector remains unfilled because it would require too much system CPU time to rearrange the file organisation every time there is a change. When File A is extended it completely fills the first four sectors and the remainder of the extended file is stored in all of Sector 8 and part of Sector 9. Sector 4 will only be used again if a small file is created or if the disk fills up, when it might store the first part of a longer file.

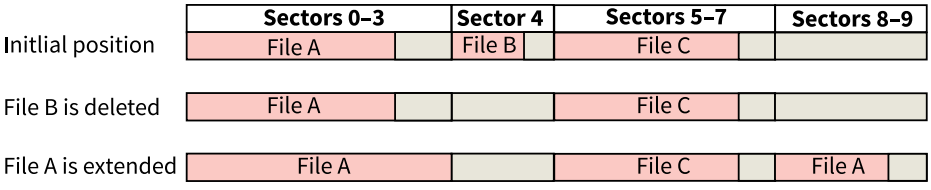


Figure 8.01 File fragmentation on a hard disk

A defragmenter utility program reorganises the file storage to return it to a state where all files are stored in one block across a sequence of sectors. For a large disk this will take some time. It will be impossible if the disk is too full because of the lack of working space for the rearrangement.

TASK 8.02

If you have never used a disk defragmenter or disk repair utility program, can you get access to a system where you can use one? If so, note the changes that are carried out and recorded by the utility program.

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:

- establish a schedule for backups
- only create a new backup file when there has been a change.

Question 8.02

In the systems that you use, the technical staff will have made provision for backup. Can you find out what procedures are followed and what hardware is used for this?

File compression

A file compression utility program can be used regularly by an operating system to minimise hard disk storage requirements. Whether or not the operating system does this, a user can still choose to install a suitable program. However, as was discussed in Chapter 1 (Section 1.07) compression is most important when transmitting data. In particular, it makes sense to compress (or 'zip') a file before attaching it to an email.

Virus checker

A virus-checking program should be installed as a permanent facility to protect a computer system. In an ideal world, it would only need to be used to scan a file when the file initially entered the system. Unfortunately, this ideal state can never be realised. When a new virus comes along there is a delay before it is recognised 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.

8.04 Program libraries

The ‘programs’ in a program library are usually subroutines created to carry out particular tasks. A programmer can use these within their own programs.

All newly developed programs are likely to contain errors, which only become apparent as the programs are tested or used. It saves a programmer a lot of time and trouble to be able to include already tried and tested subroutines taken from a program library.

The most obvious examples of library routines are the built-in functions available for use when programming in a particular language. Examples of these are discussed in Chapter 14 (Section 14.07). Another example is the collection of over 1600 procedures for mathematical and statistics processing available from the Numerical Algorithms Group (NAG) library. This organisation has been creating routines since 1971 and they are universally accepted as being as reliable as software ever can be.

In Section 8.05, we will discuss the methods available for translation of source code. For now, we simply need an overview of what happens. The source code is written in a programming language of choice. If a compiler is used for the translation and no errors are found, the compiler produces object code (machine code). This code cannot be executed by itself. Instead it has to be linked with the code for any subroutines used by it. It is possible to carry out the linking before loading the full code into memory and running it.

There is a major disadvantage in linking library routines into the executable code. This is because every program using a routine has to have its own copy. This increases the storage space requirement for the executable file. It also increases memory usage when more than one process uses the routine.

The alternative is to use a routine from a dynamic linked library (DLL). When a DLL routine is available the executable code just requires a small piece of code to be included. This allows it to link to the routine, which is stored separately in memory, when execution of the program needs it. Many processes can be linked to the same routine. This has the advantage that the executable files for all programs need less storage space. Memory requirement is also minimised. Another advantage is that if a new version of the routine becomes available it can be loaded into memory so that any program using it is automatically upgraded.

The main disadvantage of using a DLL is that the program is relying on the routine being available and performing the expected function. If for some reason the DLL becomes corrupted or a new version has bugs not yet discovered the program will fail or produce an erroneous result. The user running the program will find it difficult to establish what needs to be done to get the program to run without error.

8.05 Language translators

The use of an assembler for translating a program written in assembly language has been discussed in Chapter 6 (Sections 6.02, 6.03 & 6.04). This chapter will introduce the translators that are used to translate a program written in a high-level procedural language.

Compilers and interpreters

The starting point for using either a compiler or an interpreter is a file containing source code, which is a program written in a high-level language.

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 analysed.
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.

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 analysed.
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. Alternatively, the object code is stored and the program is executed later with no involvement of the compiler.

Discussion Point:

What type of facility for language translation are you being provided with? Does your experience of using it match what has been described here?

The advantages and disadvantages to a programmer of creating interpreted or compiled programs.

- An interpreter has advantages when a program is being developed because errors can be identified as they occur and corrected immediately without having to wait for the whole of the source code to be read and analysed.
- An interpreter has a disadvantage in that during a particular execution of the program, parts of the

code which contain syntax errors may not be accessed so if errors are still present, they are not discovered until later.

- An interpreter has a disadvantage when a program is error free and is distributed to users because the source code has to be sent to each user.
- A compiler has the advantage that an executable file can be distributed to users, so the users have no access to the source code.

The advantages and disadvantages to the user of interpreted or compiled programs.

- For an interpreted program, the interpreter and the source code have to be available each time that an error-free program is run.
- For a compiled program, only the object code has to be available each time that an error-free program is run.
- Compiled object code will provide faster execution than is possible for an interpreted program.
- Compiled object code is less secure because it could contain a virus.

Whether an interpreter or a compiler is used, a program can only be run on a particular computer with a particular processor if the interpreter or compiler program has been written for that processor.

If there is an option available the choice of an interpreter is justified when a program is being developed because:

- one error in a program can lead to several other errors occurring
- an interpreter can detect and correct an early error so limiting subsequent ones
- the debugging facilities provided in association with the interpreter speed this process.

The choice of a compiler is justified when the programmer is confident that the program is as near error-free as possible because:

- an executable file can be created
- this can be distributed for general use
- execution of the program will be faster than if an interpreter were used.

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.

8.06 Features found in a typical Integrated Development Environment (IDE)

Whatever language is used for writing source code and whatever compiler or interpreter is being used there will be one or more IDEs available to assist the programmer. This section discusses the types of feature that should be provided by an IDE.

Prettyprinting

Prettyprint refers to the presentation of the program code typed into an editor. For example, the Python IDLE (see Figure 8.02) automatically colour-codes keywords, built-in function calls, comments, strings and the identifier in a function header. In addition, indentation is automatic.

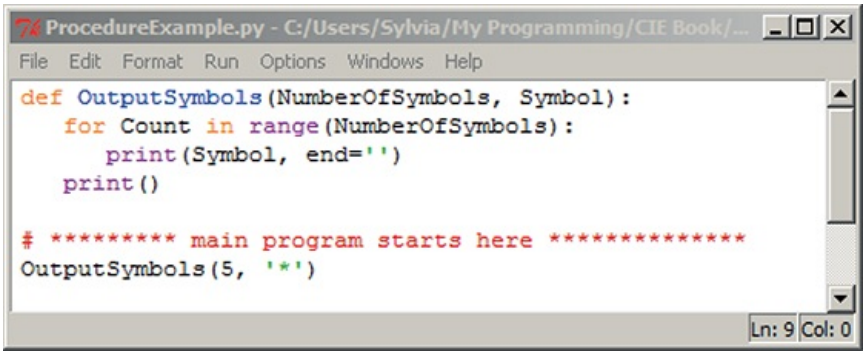


Figure 8.02 Prettyprint in the Python IDLE

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. Figure 8.03 shows an example of the Visual Studio editor responding to text typed in by the programmer.

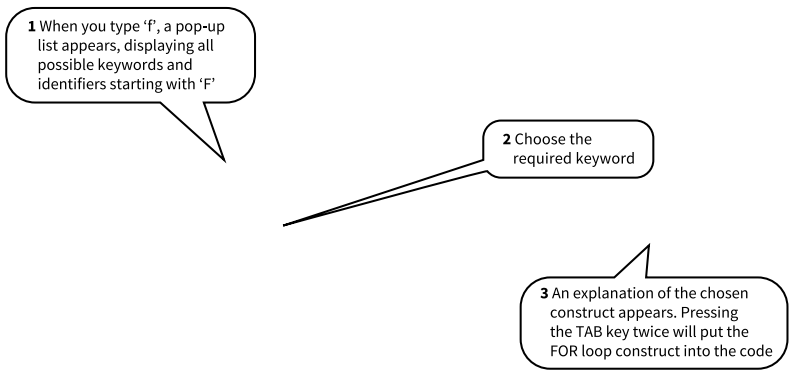


Figure 8.03 Context-sensitive prompts in the Visual Studio editor

Dynamic syntax checks

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

Figure 8.04 shows an example of the Visual Studio editor responding to a syntax error.

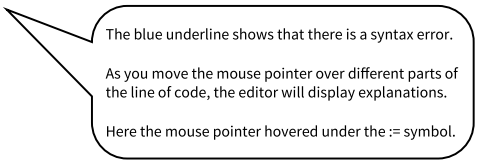


Figure 8.04 Dynamic syntax check in the Visual Studio editor

Expanding and collapsing code blocks

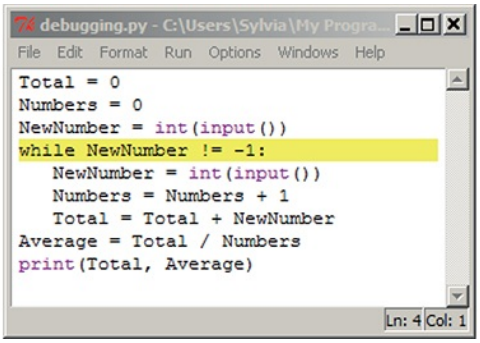
When working on program code consisting of many lines of code, it saves excessive scrolling if you can collapse blocks of statements.

Debugging

An IDE often contains features to help with **debugging**.

If a Debugger feature has been switched on it is possible to select a breakpoint. When the program starts running it will stop when it reaches the breakpoint. The program can then be stepped through, one instruction at a time. Figure 8.05 shows the windows presented to the user in the Python IDLE when this feature is being used.

a



b

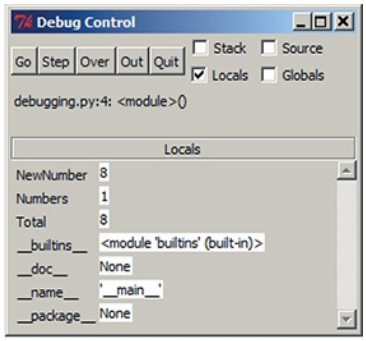


Figure 8.05 (a) A Python program showing a breakpoint; (b) the Debug Control window

TASK 8.03

Investigate the facilities in the editors you have available. If you have a choice of editors, you may like to use the editor with the most helpful facilities.

Reflection Point:

Much of the discussion in this chapter only summarises what an operating system does. Do you think it would be helpful to move on immediately to have a look at some of the content in sections 20.01 to 20.05 of Chapter 20?

Summary

- Operating system tasks can be categorised in more than one way, for example, some are for helping the user, others are for running the system.
- Utility programs include hard disk utilities, backup programs, virus checkers and file compression utilities.
- Library programs, including Dynamic Link Library (DLL) files, are available to be incorporated into programs; they are usually subroutines and are very reliable.
- A high-level language can be translated using an interpreter or a compiler.

- A Java compiler produces Java Byte Code which is interpreted by a Java Virtual Machine.
- An integrated Development Environment (IDE) contains many features that provide support for a programmer when a program is being written and when it is being corrected.

Exam-style Questions

1 **a** One of the reasons for having an operating system is to provide a user interface to a computer system.

i Name **two** different types of interface that an operating system should provide. [2]

ii Identify for each type of interface a device that could be used to enter data. [2]

b Identify and explain briefly **three** other management tasks carried out by an operating system. [6]

2 **a** A PC operating system will make available to a user a number of utility programs.

i Identify **two** utility programs that might be used to deal with a hard disk problem. [2]

ii For each of these utility programs explain why it might be needed and explain what it does.

iii Identify **two** other utility programs for a PC user. [2] [5]

b Library programs are made available for programmers.

i Explain why a programmer should use library programs. [3]

ii Identify **two** examples of a library program. [2]

3 Assemblers, compilers and interpreters are examples of translation programs.

a State the difference between an assembler and a compiler or interpreter. [1]

b A programmer can choose to use an interpreter or a compiler.

i State **three** differences between how an interpreter works and how a compiler works. [3]

ii Discuss the advantages and disadvantages of an interpreter compared to a compiler. [4]

iii If a programmer chooses Java, a special approach is used. Identify **one** feature of this special approach. [1]

4 **a** Explain the meaning of the following terms:

Prettyprinting

Context-sensitive prompt

Dynamic syntax check

Debugging [8]

b Describe the features you would expect a debugger to provide. [4]

5 Before it is used, a hard disk is formatted using disk formatter software.

a Explain why formatting is needed. [2]

b Eventually, the performance of the hard disk deteriorates.

Name **three** other utility programs that might be required. State why each is needed. [6]

Cambridge International AS & A level Computer Science 9608 paper 12 Q10 November 2015

6 A programmer is writing a program that includes code from a program library.

a Describe **two** benefits to the programmer of using one or more library routines. [4]

b The programmer decides to use a Dynamic Link Library (DLL) file.

i Describe **two** benefits of using DLL files. [4]

ii State **one** drawback of using DLL files. [2]

Cambridge international AS & A Level Computer Science 9608 paper 12 Q8 November 2016