Operating Systems (Client)

Lecture 1
Fundamental Concepts

Higher Cert. in Science in Computing (IT) – BN002
Bachelor of Science in Computing (IT) – BN013
Bachelor of Science (Hons) in Computing – BN104

Dr. Kevin Farrell

itb
Institute of Technology
Blanchardstown
Institiúid Teicneolaíochta
Baile Bhlainséir

Contents

Figures

License

Feedback

Constructive comments and suggestions on this document are welcome. Please direct them to:

kevin.farrell@itb.ie

Acknowledgements

Thanks in particular go to the Dr. Anthony Keane, whose lectures this material was partly based on.

Modifications and updates

Version	Date	Description of Change
1.0	1st June 2005	First published edition.
1.1	10th August 2005	Minor formatting changes

1. Overview

1.1. Lecture Summary

In this Lecture, we introduce the fundamental concepts of operating systems (OS). The first part of the Lecture aims to address *four* key questions:

- What is an operating system?
- What do operating systems do?
- What resources need to be managed by operating systems?
- What are the components of an operating system?

In the second part of the Lecture, we examine a number of common *types* of operating systems, in addition to giving a brief history of the evolution of operating systems.

1.2. Learning Outcomes

After successfully completing this Lecture you should be able to:

- describe what an operating system is.
- describe what an operating system does.
- describe the types of resources managed by operating systems
- describe the main components of an operating system.
- explain the function of the *four* OS managers.

1.3. Reading

Material for this lecture was gathered from very many different sources and none! Some good core texts are the following:

- “Operating System incorporating Windows and UNIX”, Colin Ritchie.

- "Operating Systems", William Stallings, Prentice Hall, (4th Edition, 2000)
- "Modern Operating Systems", Andrew Tannenbaum, Prentice Hall, (2nd Edition, 2001).
- "Operating System Concepts", Silberschatz, Galvin & Gagne, John Wiley & Sons, (6th Edition, 2003).

1.4. Suggested Time Management

This Lecture should take between 3 and 5 hours of your study time:

- Lecture Content: 2 hours
- Further reading: 2 hours

Since different authors have different approaches to operating systems, it is a good idea at this early stage to read the introductory chapters from a number of different textbooks.

1.5. Typographical Conventions

Throughout this Operating Systems Module, I have tried to use uniform typographical conventions; the aim being to improve readability, and lend understanding:

- Key/Technical Terms: **Bold Underline** for eg: **Multiprogramming**
- Emphasis: *Italics*
- Command names: **Bold**
- Filenames/Paths: `Courier` for eg: `/home/kevin/cv.doc`

2. Introduction

To understand the operating system is to understand the workings of the entire computer system because it is the operating system that manages each piece of the hardware and software. This Lecture is divided into two parts. The first part introduces the basic concepts of operating systems. In it, we aim to address a number of questions:

- What is an operating system?
- What do operating systems do?
- What resources need to be managed by operating systems?
- What are the components of an operating system?
- What are the major Issues associated with designing an operating system?

The second part of this Lecture gives a brief history of various different types of operating systems which have existed. We examine how they have evolved to the types of operating systems with which we are familiar today. By examining this evolution, we hope to gain insights into how operating systems will develop in the future.

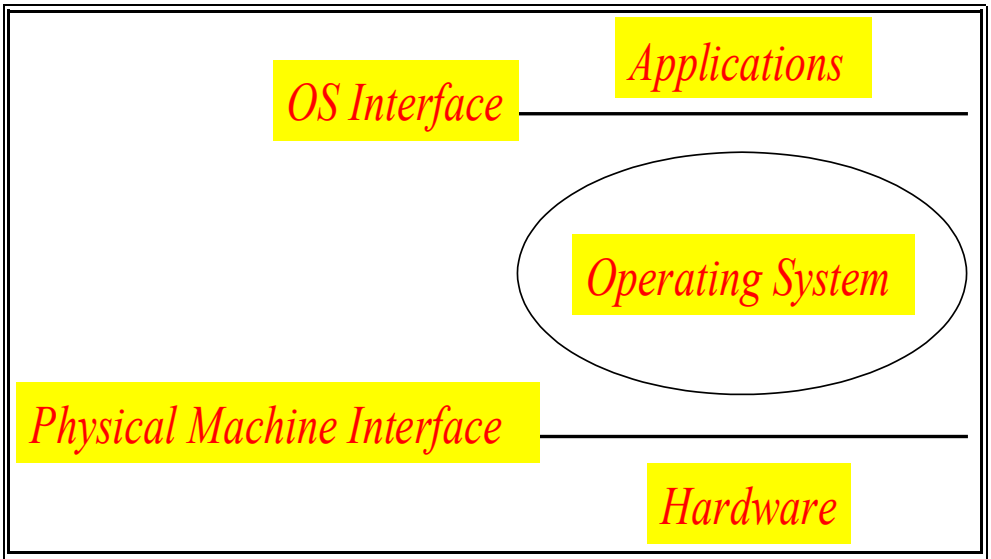

Figure 2.1. Relationship of the Operating System to Hardware and Software (Applications).

3. What is an Operating System?

In simple terms, an operating system is the "*executive manager*" of the computer system. It controls every file, every device, every section of memory and every nanosecond of processing time. It also controls who can use the system and how. A more formal definition might be as follows:

> *Definition 3.1*
>
> An operating system (OS) provides a virtual machine on top of the hardware which is more convenient than the raw hardware interface itself.

Figure 2.1 shows a schematic drawing indicating the relationship of the operating system to hardware and software (applications). Each of these has an interface with the OS; the **Physical Machine Interface** lies between the Hardware and the OS, and **OS Interface** lies between the Applications and the OS.

For you, as an application developer or programmer, an OS is "all of the code you don't write!" The applications you write will need to use the hardware. Instead of your having to write all of the required code to control the hardware, a team of software designers and programmers have created a general piece of software to do this for you; namely, the operating system! Your applications therefore must interact with the OS. This interaction takes place at the OS interface. The OS in turn controls all of the aspects of the hardware which your application requires. There are several advantages to using this approach to control a computer system, namely that it is,

- Simpler
- More reliable
- More secure
- More portable
- More efficient

4. What do Operating Systems do?

We can now summarise briefly in the next subsections what an OS actually does.

4.1. Manages physical and virtual resources.

Examples of physical resources are the hardware. An example of a virtual resource is a printer queue.

4.2. Provide users with a well-behaved environment

The system should behave the same way each time it is turned on, and each time the user performs the same set of actions.

4.3. Define and Implement Logical Resources and Rules

The OS should define a set of logical resources (objects) and a set of well-defined operations on those resources (i.e. an interface to those objects). For example, in UNIX all devices are represented as special files. These are logical resources. In order for a user to use a device, that user must use the logical resource of the device; i.e. the user must interact/interface, in a particular way, with the special file, which represents the device. These operations often consist of reading from and writing to the special file representing the device.

4.4. Provide *mechanisms* and *policies* for the control of resources

It is important to distinguish between **mechanisms** and **policies**. Mechanisms determine *how* something will be done; policies decide *what* will be done. The separation of mechanism and policy increases flexibility. Policies are likely to change from place to place or time to time. In the worst case, every change in a policy would require a change in the underlying mechanism. General mechanisms are more desir-

able, because a policy change would require the modification of only some system parameters or tables.

4.5. Control how different users and programs interact

Nearly all contemporary operating systems in general use allow multiple users to use the system and multiple programs to be run concurrently. Therefore, the operating system needs to have the facility to control how different users and programs interact.

5. What Resources need to be Managed?

By broadening the von Neumann view of a computing system, we can summarise the resources that need to be managed by the OS:

- The CPU(s)
- Memory
- Storage Devices (disks, tapes, etc.)
- Input Devices (keyboard, mouse, cameras, etc.)
- Output Devices (printers, displays, speakers, etc.)
- Networks

At this stage, these need little explanation! However, in later Lectures, we will expand on *how* these resources are managed.

6. Components of an Operating System

Referring to Figure 2.1, we can propose a logical structure of an operating system, which represents the OS in more detail, by naming some specific essential services. Figure 6.1 shows the resulting schematic.

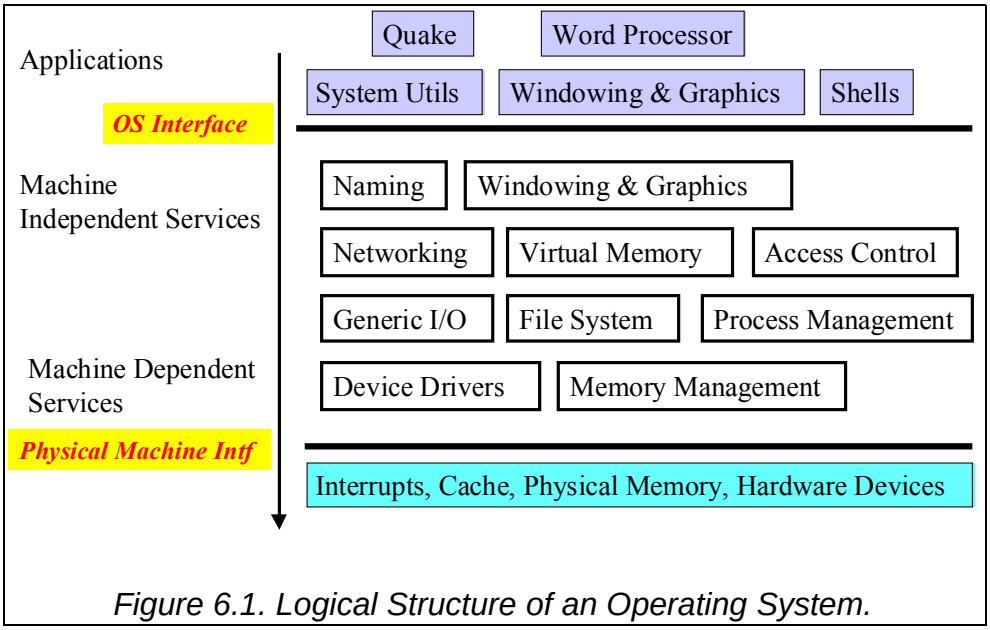

Figure 6.1. Logical Structure of an Operating System.

As with Figure 2.1, the OS lies between the OS Interface and the Physical Machine Interface. Each service in the system has its place according to how dependent it is on the physical hardware structure. In Figure 6.1, we can see that services placed near the bottom of the OS, are more machine dependent than those near the top.

One interesting point to note from this figure, is the inclusion of “Windowing & Graphics” in both the OS, and outside the OS, in applications. Some operating systems incorporate windowing and graphics into the OS. Generally, this increases performance, but with the disadvantage that the system is more complex, which can result in less stability. One example of such a system is Microsoft Windows 9x/NT family. Other operating systems, such as GNU/Linux[1] (Linux) for example, do not incorporate the core windowing and graphics functionality into the OS. Instead, the graphical user interface (GUI) is provided by a separate application, the **X Windows System**. This has a number of advantages. Firstly, the OS is less complex, and one would expect it to be more stable as a result (which it is!). Secondly, one can run the OS without a GUI, leaving more memory for other applications, making the OS ideally suited to being used as a server.

1 GNU (= GNU's Not UNIX) software, consisting of essential software and system tools, originates from the Free Software Foundation. Without it, Linux would be just an operating system kernel, which nobody could use. Hence, when people talk of the Linux OS, they generally mean the Linux kernel + GNU software. Together, these make a useful operating system.

Having determined what services should be contained in an OS, how does one go about designing one? Traditionally, operating system design has involved dividing the system into several components, each with different responsibilities. Figure 6.2 shows a schematic of these typical components, and indicates which components interact with each other.

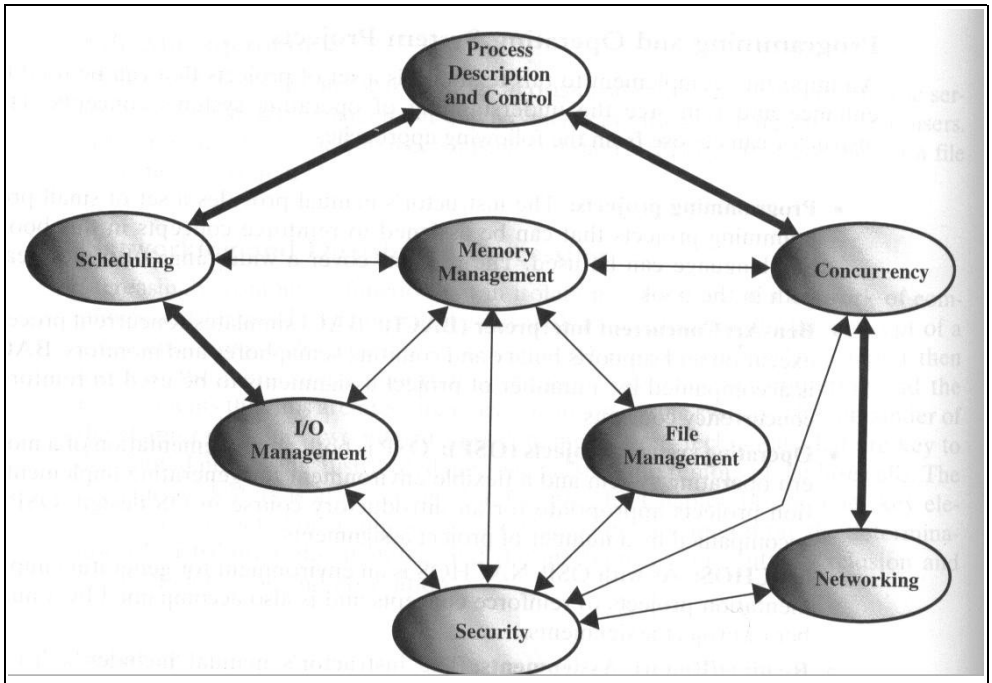

Figure 6.2: Operating system design consists of division of the system into several interacting components as shown in this schematic.

Therefore, it should not come as a surprise that the study of operating systems is made up of several basic topics focussing on the above components. We describe these briefly in the next subsections.

6.1. Process Description and Control

Operating systems must allocate resources to processes, enable processes to share and exchange information, protect the resources of each process from other processes and enable synchronisation among processes.
Subtopics are **scheduling** and **concurrency**.

6.2. Memory Management

The memory of a typical computer is occupied by a wide range of different objects such as OS code and data, user program code and data, video storage space and so on. This topic looks at how the OS manages the memory space.
Subtopics are **process loading**, **process swapping**, **memory allocation** methods, **virtual memory**, **protection** and **sharing**.

6.3. I/O Management

This is the least satisfactory aspect of OS design due to the wide variety of I/O devices and their applications. Other problems arise due to the rapid speed of a processor and the relative slowness of an I/O device.
Subtopics are **programmed I/O**, **interrupt driven I/O**, **direct memory access (DMA)**, **buffering**, **disk scheduling** and **disk caching.**

6.4. File Management

For most applications, the file is the central element. Files are used as a means of long-term storage using a secondary storage device like a hard disk. File management involves the study of alternative ways to organise files, the services provided by the OS to deal with files, and techniques used to improve disk performance. A subtopic is **file security**.

There are a number of subcomponents/subsystems shown in Figure 6.2, which are required for the system to work, which we have not mentioned yet. We will devote an entire Lecture later to one of these, namely **scheduling**.

7. Operating System Managers

So we see that an operating system has *four* primary "managers", which are designed to manage specific resource-types (see Figure 7.1):

- Processor Manager
- Memory Manager
- Device Manager (sometimes called the I/O Manager)
- File Manager

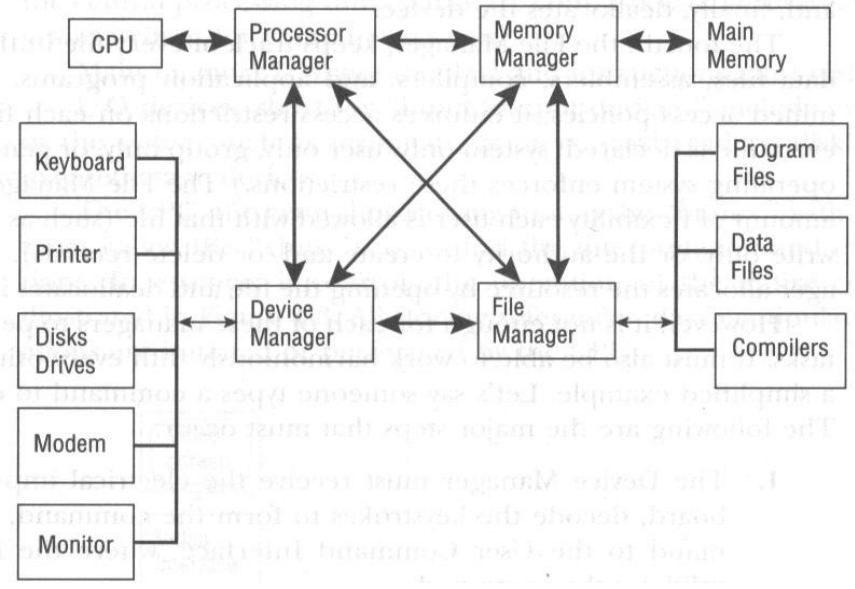

Figure 7.1: Operating system design consists of four managers, which manage specific resource-types.

Each manager works closely with all of the others and performs its unique role regardless of which specific operating system is being discussed. Regardless of the size or configuration of the system, each of the subsystem managers must perform these following tasks:

- monitor its resources continuously
- enforce the policies that determine who gets what, when and how much
- allocate the resource when it is appropriate
- de-allocate the resource—reclaim it—when appropriate

An example of a resource is the Floppy Drive. When you want to see information about what files, which are on a floppy disk, you insert the disk into the floppy drive, and change to that drive; the “A:” drive. You will then be able to see the files on the floppy disk by typing the command dir.

Now that is mighty resourceful! Imagine you had to write the code that carries out the communication with the floppy drive, which instructs it to retrieve information from the inserted floppy disk and then display it on screen. Even on a simple machine, it would take quite a bit of code to do this. We can say that the floppy drive and floppy disk are **physical** pieces of hardware (physical resources), and the “A:” that represents it is an **abstraction** of a floppy disk in the floppy drive (logical resource).

An OS does all this for you. It gives you resources that make your life easier. In order to use an operating system effectively as a programmer, and learn how best to write your own code for applications, it is important to know *how* an OS works.

7.1. Interaction between OS Managers

It is not enough for each of the managers to perform its individual tasks; it must also work harmoniously with every other manager. Here is a simplified example.

Let’s say someone types a command to execute a program. The following are the major steps that must occur:

1. The **Device Manager** must receive the electrical impulses from the keyboard, decode the keystrokes to form the command and send the command to the User Command Interface, where the **Processor Manager** (or **Process Manager**) validates the command.
2. The Processor Manager then sends an acknowledgement message to the display on the video monitor so the user will realise the command has been sent.
3. When the Process Manager receives the command, it determines whether the program must be retrieved from storage or is already in memory and then notifies the appropriate manager.
4. If the program is in storage, the **File Manager** must calculate its exact location on the disk, pass this information to the Device Manager which will retrieve and send the program on to the **Memory Manager**, which must find space for it and record its exact location in memory.
5. Once the program is in memory, the Memory Manager must track its location and progress as the Process Manager executes it.
6. When the program has finished executing, it must send a “finished” message back to the Process Manager.
7. Finally, the Processor Manager must forward the “finished” message back to the Device Manager, which displays it on the video monitor for the user to see.

Note: this is an oversimplified example of a complex operation used to illustrate the precision and co-operation required by the operating system and its parts.

8. System Calls

System calls provide the interface between a process and the operating system. They are the mechanism by which an application calls the system to ask it to operate the hardware on its behalf. We will discuss the details of system calls in a later Lecture. But for the moment, here is an example of the types of system calls that go on in a process:

Process control:

- end, abort
- load, execute
- create process, terminate process
- get/set process attributes
- wait for time
- wait for event, signal event
- allocate and free memory

File manipulation:

- create file, delete file
- open, close
- read, write, reposition
- get/set file attributes
- Device manipulation
- request/release device
- read, write, reposition
- get/set device attribute
- logically attach/detach devices
- Information maintenance
- get/set time/date
- get/set system data
- get process, file or device attributes
- set process, file or device attributes

Communications:

- create, delete communication connection
- send/receive messages
- transfer status information
- attach/detach remote devices

9. Computer System Types

The very first PC's had no operating system. They were very basic. You were able to flip switches to change the value of inputs and the instruction to be executed. At the time, Computer Scientists realised that the system had potential, and worked on ways of improving it. Today's PCs have monitors, keyboards; file structures, graphics cards, ram, hard drives and many types of hardware. As already mentioned, to get them all to work together, you need an operating system. An operating system is all the software needed to talk to the hardware (**low level**) and all the software needed to support applications (**high level**) and everything in between.

To appreciate the role of the operating system, we need to define the essential aspects of the computer system's hardware, the physical machine and its electronic components.

While most operating systems behave in the same way inside the computer circuitry, the physical appearance of the computer and the *User Command Interface* is often quite different, e.g. MS-DOS has a text based command line interface and Windows has a graphical user interface (GUI). Computers started out with a User **Command Line Interface** (CLI). As computers became more powerful GUI's were added for convenience. That convenience though comes at a cost of increasing complexity of software and the need for more hardware resources, like memory.

Let's have a look at some different types of computer systems:

- ***Mainframe***

  The IBM 360 model 30 (the smallest computer in this series), required an air conditioned room, about 18 foot square space to house the CPU, console, printer, punchcard reader and a key punch machine. The CPU was 5ft high and 6ft wide, and had an internal memory of 64K and cost $200,000.

- ***Mini-Computer***

  Smaller in size and capacity than mainframes and targeted at businesses.

Digital Equipment Corporation (DEC) marketed the Peripheral Device Processor 8 (PDP-8) at $18,000 in the early 1970s.

- ***Micro-Computer***

  Early personal computers (PCs) had 64K of memory but were cheaper than mini-computers. A modern PC has 256MB of memory, 100GB of secondary memory (or secondary storage) and a 4GHz 64bit processor. Networked computers today give the same if not better performances than a mainframe of several years ago.

Rapid advances in technology have blurred the historical distinguishing characteristics of machines – computers today are compared by their processing power and the most powerful machines have several processors at work simultaneously.

10. Types of Operating Systems

We now give a brief description of the different *types* of operating systems, which have existed, some of which have persisted to this day.

10.1. Batch Systems

Early computer systems relied on punched cards or tape for input. A job was entered by assembling the cards into a “deck”, and running the entire deck of cards through a card reader as a group – a “batch”.

In present day systems, the term batch refers to a series of jobs being processed together without interaction, e.g. Payroll.

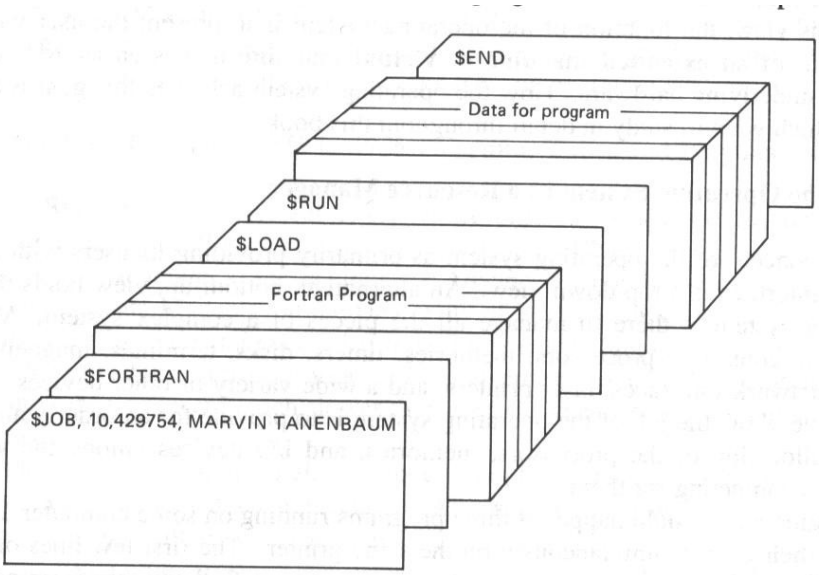

Figure 10.1: . Schematic showing batch system program cards. Control cards are indicated by a "$" symbol.

10.2. Multiprogramming Batch systems

This type of OS kept multiple runnable jobs loaded in memory, and overlapped I/O processing of a job with computation of another. This provided increased utilisation:

- Benefited from I/O devices that can operate asynchronously
- required the use of interrupts and DMA (Direct Memory Access)
- Optimised for **throughput** at the cost of **response time**

10.3. Hybrid Systems

These are a combination of batch and interactive type systems. The system appears to be interactive since individuals can access the system via terminals and get fast response, but the system actually accepts and runs batch programs in the background when the interactive load is low. Many large computer systems are hybrid in nature.

10.4. Interactive Systems/Time-sharing Systems

The next development in operating systems was to allow the user to type in the commands via a typewriter like terminal. i.e. **Time-sharing** supported *interactive* computer use:

- Each user connects to a central machine through a cheap terminal; feels as if she has the entire machine
- Based on time-slicing: dividing CPU equally among the users
- Permitted active viewing, editing, debugging, participation of users in the execution process
- Security mechanisms required to isolate users from each other
- Requires memory protection hardware for isolation
- Optimizes for *response time* at the cost of *throughput*

This allowed a faster turnaround measured in seconds or minutes, depending on the number of active users, e.g. debugging a program.

10.5. Distributed systems

These types of systems consist of computers connected by wires. They have no shared memory or clock (i.e. each CPU only has access to the memory in its computer, and each CPU “ticks” at its own rate). They have a number of benefits, but also introduce some extra complexities, which need to be solved:

- Require support for communication between parts of a job or different jobs
- Inter-process communication (IPC)
- facilitate use of geographically distributed resources
- Users benefit from sharing of distributed resources, hardware and software (Resource utilisation and access)
- Permit some parallelism, but speeding up execution of processes is not the primary objective

10.6. Parallel Operating Systems

Objective: Support parallel applications wishing to get speed up of computationally complex tasks. Needs basic primitives (simple rules) for dividing one task into multiple parallel activities:

- OS must support efficient communication between those activities
- OS must support synchronization of activities to coordinate sharing of information

It is now common to use networks (*clusters*) of high-performance ordinary PCs/workstations as a parallel computer. Hence we use the term **COTS** (**Commodity Off The Shelf**) technology. Most clusters run the Linux operating system combined with some essential message-passing libraries. These types of systems are now quite common in certain fields:

- Military and Aerospace: nuclear weapons safety assessment, aircraft design
- Scientific: weather prediction, oil exploration, vehicular traffic prediction
- Film Industry: special effects and animation

10.7. Real-Time Systems (RTOS)

These are used for “time critical” environment where data must be processed extremely quickly because the output will influence immediate decisions; i.e. need to cope with rigid time constraints. There are two types:

- Hard real-time
  OS guarantees that applications *will* meet their deadlines
  Examples: CD player, TCAS, health monitors, factory control
- Soft real-time
  OS provides prioritisation, on a best-effort basis. No deadline guarantees, but bounded delays
  Examples: most electronic appliances

Real-time means “predictable”. Some RTOS are very fast, others are NOT necessarily fast.

11. A Brief History of Operating Systems and Associated Hardware

Here follows a brief overview of the different operating systems and their associated hardware platforms. Figure 11.1 is a graphic showing a history of operating systems with associated software tools and hardware-types.

11.1. First Generation (1940 – 1955)

The early computers were large machines due mainly to vacuum tubes. Each computer was unique in structure and purpose, mathematical, scientific or military applications. Computer operated from a console and the computer would have to be reserved to run a program. The programs did everything, for example a typical program would include all the instructions needed to perform the tasks required, e.g. control of the card reader, the CPU and the output device.

- *card reader* – where to begin, how to interpret the data, where to end.
- *CPU* – how and where to store the instructions and data in memory, what to calculate and where to send the output.
- *output device* – when to begin, how to print out the data, how to format the page and where to end.

For a programmer to debug a program, he would stop the processor, read the contents of each register, make the corrections in memory locations and resume operation.
In time computer hardware/software became more standard:

- assemblers/compilers

- basic OS with macros, libraries, standard subroutines.
- device driver subroutines standardised I/O devices.
- Magnetic storage technology

11.2. Second Generation (1955 – 1965)

New computers were developed to meet the demands of business. Improvements included the use of operators to run the computers, and job scheduling to increase CPU usage. This led to the development of Job Control Language (JPL).

Factors to improve the performance of the CPU were:

- speed of I/O devices became faster
- blocking
- control unit -> buffering
- spooling
- timer interrupts

Processing was still run in serial batch mode.

11.3. Third Generation (1965 – 1975)

There are several.

11.3.1. Multiprogramming systems

The term muliprogramming refers to a system, which allows a number of programs to be executed over a *period* of time. The term “concurrent” (rather than simultaneous) execution is used to describe the life-cycle of programs in such systems. Essentially, programs are executed one after another[2]. There are two types of multiprogramming:

- passive – program hogs the CPU until it is finished.
- active – time slices for each program.

2 The term “concurrent” (rather than simultaneous) execution is used to describe multiprogramming systems. These are systems, where a number of processes are executed over a *period* of time.

Other associated terms are:

- **Program scheduling** – loading programs into memory according to priority/memory requirements
- **Job scheduling** – how to cope when two programs want same device at the same time.

11.3.2. Core Resident Operating Systems

- Post-Third Generation (Late 1970s)
- firmware
- virtual memory – solution to memory limitation requirements “roll in/roll out”
- multiprocessing

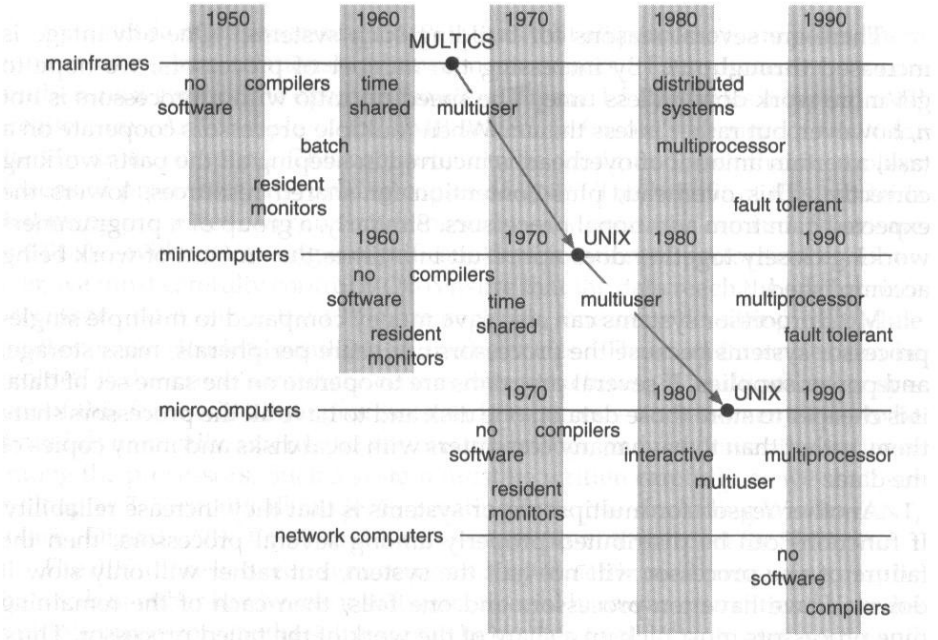

Figure 11.1: A history of operating systems with associated software tools and hardware-types.

12. Exercises

1. Fill in the blanks. (The possible choices are Process, Memory, File and Device).

   Q. What are the steps involved in the execution of a program from the command line?

   A. The following are the major steps that must occur:

   The _____ Manager must receive the electrical impulses from the keyboard, decode the keystrokes to form the command and send the command to the User Command Interface, where the _____Manager validates the command.

   The _____ Manager then sends an acknowledgement message to the display on the video monitor so the user will realise the command has been sent.

   When the _____ Manager receives the command, it determines whether the program must be retrieved from storage or is already in memory and then notifies the appropriate manager.

   If the program is in storage, the _____ Manager must calculate its exact loca-

tion on the disk, pass this information to the _____ Manager which will retrieve and send the program on to the _____ Manager, which must find space for it and record its exact location in memory.

Once the program is in memory, the _____ Manager must track its location and progress as it is executed by the _____ Manager.

When the program has finished executing, it must send a “finished” message back to the _____ Manager.

Finally, the _____ Manager must forward the “finished” message back to the _____ Manager, which displays it on the video monitor for the user to see.

2. What system calls do you think would be used by a program that wants to read data from one file and write to another file?
3. Distinguish between *soft real time operating systems* and *hard real time operating systems.* Give one example in each case of where such a type of real time system is used.