1 Introduction to Operating Systems

1.1 INTRODUCTION

The power of computing has changed the lives of common people in the last two decades. Computer systems have also been through a lot of change. In this information age, everybody is concerned with the computer and its speed; whether you are surfing the web, booking tickets through the Internet, or accessing online banking. Mobile phones have also doubled up as mini computers known as *smartphones*. Everyone needs a computer system that will meet their requirements at a great speed. Whether you are working on a stand-alone system or transferring your files on the network, the job is done with not much effort. Have you ever thought how all this works? How does a machine know whether you are working on a stand-alone system or on the network or a distributed system? Is the machine capable of knowing all your needs? Do you think that the hardware performs all these functionalities for you? The fact is that the hardware cannot perform on its own.

There was a time in the history of computer systems when every work was done manually. At that time we were very close to the machine. But it had a lot of problems and there was no efficiency in working as we get today. Therefore, a software was designed, which worked on the hardware to relieve a general user from the machine view of the system. This software did all the functionalities that need to be performed on the hardware on behalf of the user. This software that operates the computer system is known as *operating system* (OS). It acts as a layer between the user and the hardware. It provides a friendly environment for a user. In the absence of an OS, a user would have had to be aware of every configuration of the hardware and how to interact with the devices. Thanks to the OS, the user does not need to worry about hardware or interact with the different devices. The OS, therefore, works in the background without letting us know who is doing the job.

The operating system has also been through a lot of changes in the past. In fact, with the advancement and technological changes in the computer architecture, OSs have also been evolved in parallel. The improvements in computer system have always impacted the structure of the OS. Sometimes, there is a need to modify the hardware as there is demand from the OS's designers. The design motivation of the OS has also been changed from time to time. There was a time when CPU was a costly resource and innovations in OSs were developed to utilize the CPU efficiently. As the CPU speed was increased with technological

Learning Objectives

After reading this chapter, the reader should be able to understand:

- The need for an Operating System (OS)
- The evolution of OSs
- Batch processing systems
- Multi-programming systems
- Multi-user systems
- Multi-tasking systems
- Multi-processor systems
- Distributed systems
- Real-time systems
- Embedded systems
- Goals of an OS
- Functions of OSs
- Generic components of OSs

advancements, it is not a major design issue. Today, the major issue is user convenience and response time for the user's multiple tasks. This is the reason Apple's Macintosh and Windows OSs were developed and today we are using their much improved versions.

The aim of this book is to have a basic understanding of the OS and know its components in detail. This book presents a detailed and systematic discussion of OSs.

1.2 THE NEED FOR OPERATING SYSTEMS

Before discussing or defining OSs, it would be better to first understand why we need OSs. Let us look at some questions that will help to understand the basic need of OSs as well as its functionalities:

- By the time we are ready to learn this subject, we must be conversant in at least one programming language. So let us find an answer to the question—while saving and running a program in file, what is the part of the computer system that allocates memory to this file?
- How are the files as a logical concept mapped to the physical disk?
- Today, we are able to open many windows at a time; who is managing all these windows despite a single processor?
- Who ensures that the CPU is not sitting idle or busy forever?
- Sometimes we see some messages like memory error or power failure, connection failure in network, paper jam in printer, etc. Who is detecting these errors and displaying error messages?
- Who manages the limited memory despite the large size of user programs?
- Our processes can also communicate and cooperate via some synchronization mechanisms. Who provides the communication and synchronization mechanisms?
- Who schedules tasks to the CPU for execution?
- What happens to a task when the CPU is already busy in processing some other task?
- Despite a single processor, it seems that many jobs are being executed in parallel. How does this happen?
- Suppose, some users are working in a LAN (local area network) with a single printer and more than one user gives a print command. How are the requests of multiple users on a single printer managed?
- Who protects one user's area from unauthorized access by another user's task?
- Why is it that sometimes our system hangs?

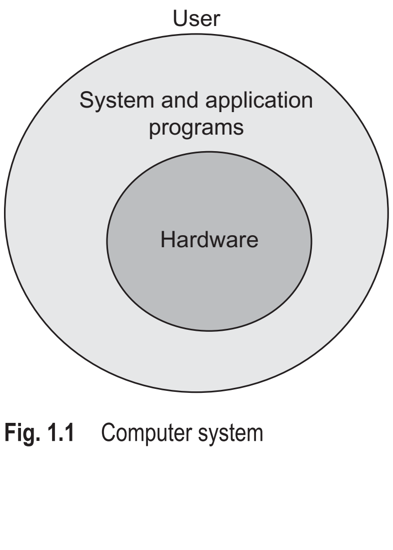


Fig. 1.1 Computer system

We always think that a computer system is a hardware and through the use of programs and other utilities, we are utilizing the system as shown in Fig. 1.1.

We work on the computer system in different ways: we write, compile, run the programs in the files or make a word file, etc. Whatever we do, we need not worry how the file is created physically, how much memory the file will take or whether the program is larger than the present RAM. So, coming back to the question, who manages and controls all the resources of the computer system?

There is a software layer between the programs and the hardware of the computer that is performing all these functions

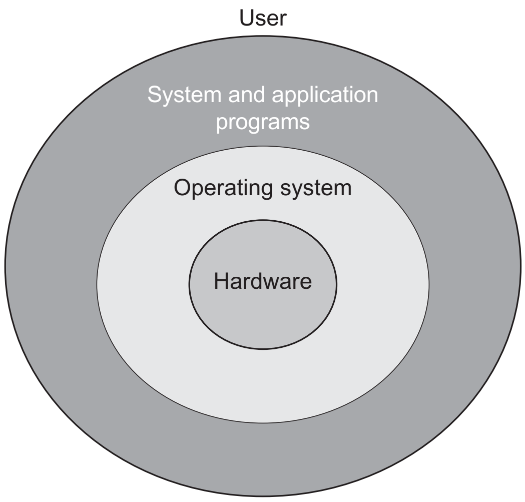


Fig. 1.2 Computer system with OS

(see Fig. 1.2). This software is known as *operating system* (OS) and is the answer to all the questions asked earlier. The OS starts functioning right from the moment we switch on the computer system and continues till we shut down the system.

From the earlier discussion, it is obvious that the OS performs the following functions:

- Presents an environment for the user so that he or she can easily work without worrying about the hardware resources, i.e., the user is not directly interacting with the resources
- Manages all the resources in an efficient manner

After understanding the role of an OS, there is still a question: Can we work without the OS? Yes, but subject to the conditions if all the functions listed earlier can be performed manually. However, this condition is not valid due to technological development in computer systems, software, and information technology. We cannot manually perform the functions of loading, saving, running, debugging, allocating memory, etc., for large-sized programs. One cannot interact with the hardware devices manually every time you need to access them. Moreover, we are living in the world of multi-tasking where users open many windows simultaneously and have a perception that they are working on all the windows in parallel. So there is no question of having just a single process for execution. The present scenario on computer systems is of multiple users with multiple processes to be executed on limited resources (see Fig. 1.3). Suppose, multiple tasks are open on a system with single CPU. The CPU time needs to be shared among all these tasks. It seems very difficult to manage CPU time among all the tasks manually. In this environment, we in fact need a software that manages all this multiplicity and controls it. This is the reason that we need OS—a software that takes care of any peripheral of the system. A software that is preparing the environment for the users such that they need not worry about the hardware details of the system. A software that is managing conflicting requests from multiple users to be executed on limited resources. A software that is protecting one user from another so that they do not interfere with one another.

It is clear now that there are certain tasks (from initialization of hardware to management and control of all the resources) that need to be performed while working on the computer system. All these tasks are put together in a single software known as the operating system. The OS can be defined in the following ways:

- A software that acts as an interface between the users and hardware of the computer system
- A software that provides a working environment for the users' applications
- A resource manager that manages the resources needed for all the applications in the background
- A software in which all common functions required to work on the computer system have been put together

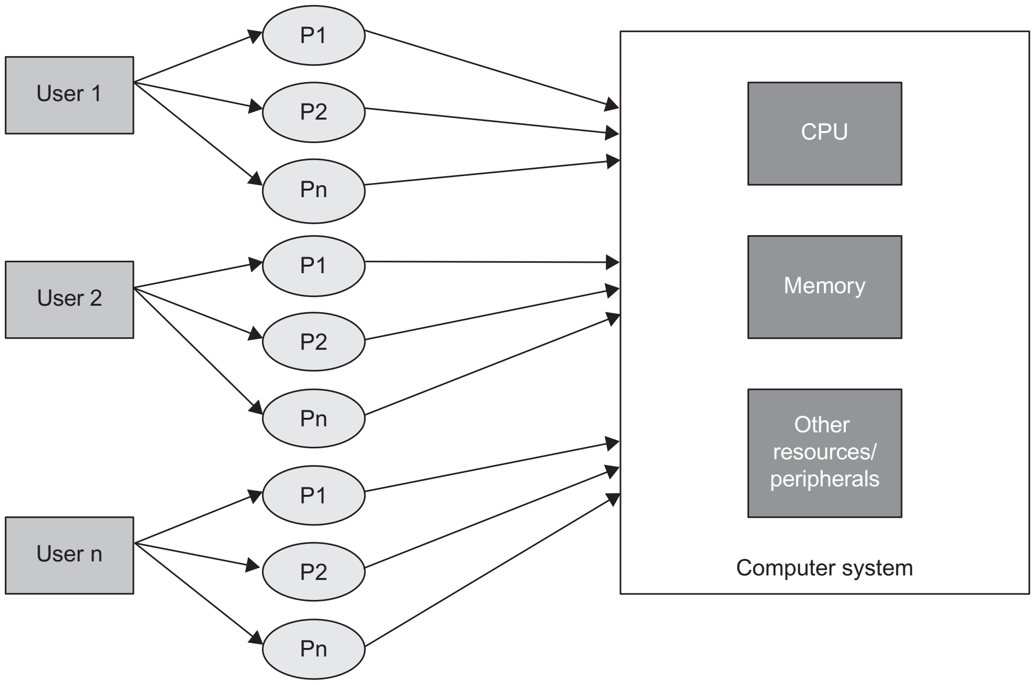


Fig. 1.3 Multiple users with multiple processes accessing the limited resources

1.3 EVOLUTION OF OPERATING SYSTEMS

It is necessary to understand the evolution of OSs. Understanding how they were developed will help us in understanding the functions of OSs we use today. There was a time when a programmer working on the computer needed to do all the activities from writing the program to loading the program into the memory, execution, debugging, and printing the output. Every activity was needed to be performed manually consuming a lot of time and resources. But today, a programmer is relieved of all other tasks of programming such as loading and linking and he or she does not care about the actual hardware details also. The programmer is only concerned with writing the program in a flexible and friendly editor and debugging it. So once we understand this, we are able to understand what the OSs mean to us today and get closer to their real requirements and objectives.

The important point in the evolution of OSs is that their development means parallel development in computer systems. As there is advancement in computer system hardware, the OSs also get updated. Sometimes, there is a demand from the OSs developers to modify the hardware. So there is influence of one on the other.

Let us have a look at how the OSs have been developed.

1.3.1 First Generation

The first generation of computer systems was the era of vacuum tubes, plug boards, and punched cards. The programmer used to interact with the machine directly through punched cards and used to write the programs in machine language. All the tasks for executing a program would be performed by the programmer only. There was no help like programming language, linker, loader, etc. Obviously, there was no operating system as we use today.

1.3.2 Second Generation

In the first generation, the programmers used to perform every task manually. There were no tools or aids for them. Therefore, in the second generation, many new softwares and hardwares were developed along with the programming languages, e.g., line printers, magnetic tapes, assemblers, linkers, loaders, and compilers for the convenience of programmers. Now the programmer would first write the program in any high-level language. To compile it, the required compiler would be loaded into the computer by mounting the compiler tape on a tape drive. Students should note that in this generation there was no hard disk to store compiler in it as we do today. The only secondary storage available was magnetic tape. But it needed to be mounted on tape drive first. After this, the program would be read through the card reader. As an output, the compiler would produce the assembly language output. This assembly language would then be assembled through an assembler. But for this purpose, it would need to mount another tape consisting of the assembler. Finally, the assembler would produce the binary object output to be loaded in the memory for execution. However, this operation of computer systems suffered from the following drawbacks:

- Since each programmer or user was allotted a fixed amount of time to execute the program on the computer system, it meant that the user would need to start all over again if there was an error at any step. This resulted in wastage of time and further delaying other users.
- Set-up time for various tasks wasted a significant amount of time of the computer system. As seen earlier, for execution of a program, there may be the following steps:
    i) Loading the compiler tape of the required compiler
    ii) Reading from card reader
    iii) Running the compiler
    iv) Unloading the compiler
    v) Loading the assembler tape
    vi) Running the assembler
    vii) Unloading the assembler tape
    viii) Loading the object program
    ix) Running the object program

For a single program execution, the loading and unloading of tapes for various purposes wasted a lot of time and delays other users.

The set-up time delay was not desirable not only for the reason that it delayed the job of users but also for the impact it had on the utilization of the CPU. In this era, the computer systems were so expensive that everyone wanted to use the system resulting in high utilization of the system. However, owing to time taken to set up, the CPU was idle most of the time. This prompted efforts towards reducing this idle time.

The solution adopted to reduce the set-up delay was that one operator was hired for loading/unloading and other tasks as listed earlier. The operator trained in loading and unloading the tapes reduced the time taken in set-up as compared to an untrained user. But, this solution was not able to reduce the actual set-up time taken by loading/unloading the tapes. Besides this, another problem was that the waiting time for the users was still high. One user had to wait while the other user executed programs on the system.

The delay in set-up time would increase further if some jobs in queue were of different requirements. For example, one job is written in FORTRAN, second in COBOL and the third is again in FORTRAN. This sequential processing of jobs would require more set-up delay as loading and unloading the tapes again and again for different jobs would be needed. If the jobs are known in advance, then we could combine these two FORTRAN jobs and send them for execution together. This would save time for loading and unloading tapes for the third job in the queue. As a solution, to reduce the waiting time of users, the jobs of users prepared with same programming language were batched together. With this solution, it was possible to execute multiple jobs instead of one and thereby saving set-up time for an individual job. For example, if there are three FORTRAN written jobs placed at different places in the queue, then a batch of these three jobs could be prepared and executed with a single set-up time as compared to three.

But in this process, the operator/user would need to check when one job would get finished and prepare the set-up for next job. So during this manual intervention, again CPU was idle. This idle time could be reduced if the switching to another job was automated instead of manual process. But this automation further required the recognition of the tasks like executing the compiler, executing the assembler, etc.

We can say that this was the turning point in the history of computer systems, when the automation for the job sequencing was conceived and thus the first operating system was born. It was thought that there would be a software called *monitor program* that would perform the automatic job sequencing. For identifying the tasks to be done for a job, the concept of *control cards* was introduced. Control cards contained the directives to tell the software which tasks to perform for a job. In this way, the software with automatic job sequencing was written that switched from one job to another. Since all the events were being controlled by the monitor now, it had to be in the memory forever. Therefore, it was called *resident monitor* and given space in the memory (see Fig. 1.4).

Resident monitor would read the control card, and load the appropriate program into the memory and run it. This process was repeated until all the cards got interpreted. Thus, the first OS in the form of resident monitor improved the execution on the computer systems. It reduced the manual intervention needed for set-up and job sequencing.

The monitor read in jobs one at a time from the card reader or tape. The job was then placed into memory in the user area and control was passed to this job. When the job was complete, the control was passed to the monitor. But the monitor needed to do several other tasks for a job. Let us look (see Fig. 1.5) at the structure of a job in the form of *job control language* (JCL), a special type of language developed to provide instructions to the monitor. The meanings of these control instructions prefixed with '$' are given in Table 1.1. At the time of executing control instructions, the control was transferred to the monitor. Otherwise, the control was transferred to the user program.

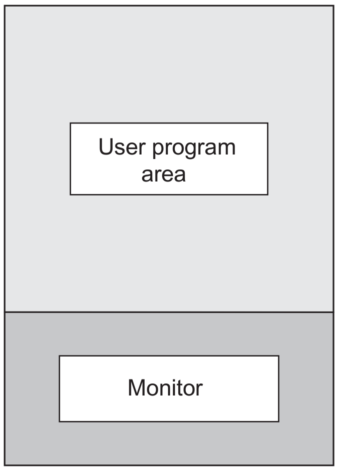


Fig. 1.4 Memory structure consisting of resident monitor

The batch systems mentioned earlier also faced one problem. In this generation, input/output (I/O) devices were mostly electro-mechanical. On the other hand, the CPU was an electronic device. While reading the input or printing the output, there was a clear mismatch between the speed of CPU and I/O devices. It was not possible to cope up with the speed of even a slower CPU that performed thousands of instructions per second. Moreover, in batch systems, it was required to execute multiple jobs with multiple control cards quickly. As a solution, it was thought that all the inputs from the card readers would be read and copied to a magnetic tape. The input would be processed in one go using this input magnetic tape. Similarly, the output, instead of being sent to the printer directly, was copied to the magnetic tape to be printed later on (see Fig. 1.6). This reduced the time taken in online processing, i.e., processing the inputs or outputs directly through the CPU. Students should note that magnetic tapes were of faster speed as compared to card readers or printers. This concept gave birth to *offline* operation. The CPU was now not constrained with the speed of slow I/O devices and was busy in processing with the I/O on tapes. The offline concept eliminated the need of processing the I/O operations through the CPU directly.

```
$JOB
$FTN
FORTRAN program
$LOAD
$RUN
$ASM
$END
```

Fig. 1.5 Format of a job in JCL

1.3.3 Third Generation

Batch systems with offline operation continued for a long time in second generation. But, still there were problems due to the nature of magnetic tapes. First, the tapes were sequential access devices. The tape needed to be rewound again and again if there was a need to write and read in between. Second, we needed to carry separate tapes for input and output as their storage capacity was low. There was no other option until the disks came into existence. The disks solved the problem faced with the tapes as disks were random access devices and of higher storage capacity as compared to tapes. Now it was possible to read or write quickly from/to any area on the disk. The cards were written directly onto the disk and read and executed the jobs from the disk. Similarly, the outputs were written directly to the same disk instead of another disk. The output was printed actually after the job is executed completely.

Table 1.1 Control instructions

$JOB	Start of a job
$END	End of a job
$ASM	Execute the assembler
$FTN	Execute the FORTRAN compiler
$LOAD	Load the object code from tape to memory
$RUN	Execute the user program

In this era, due to technological improvements in hardware, the speed of CPU became faster than the second generation. The result of this technological improvement was that CPU was more idle. The batch systems with disks improved the performance of the system, but CPU was still idle. When a job executed, it might need to have input or output, which means it needed to access devices. And the access to the devices was slow at that time as compared to the speed of CPU. So while performing I/O operation for a job, there was sufficient time for CPU to sit idle. If there had been another job, CPU would switch to it. As a solution, it was thought that instead of one job, there would now be multiple jobs in memory. Consequently, it required the memory to be partitioned for multiple jobs because till now only single job was in the memory for execution (see Fig. 1.7). This gave birth to *multiprogrammed batch systems*. It was now possible to overlap the I/O of one job with computation of another job. It means when one job was waiting for its I/O, CPU switched to another job. This was possible only with the disk systems and the method is called as *simultaneous peripheral operation online* (SPOOL). Later on, the process was famous with the name *spooling*.

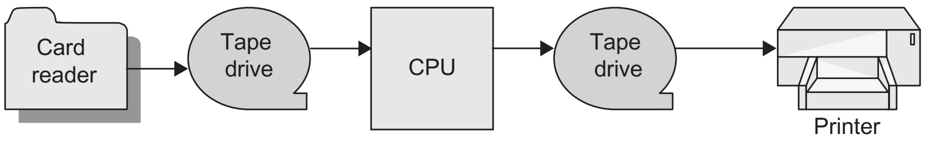


Fig. 1.6 Offline operation

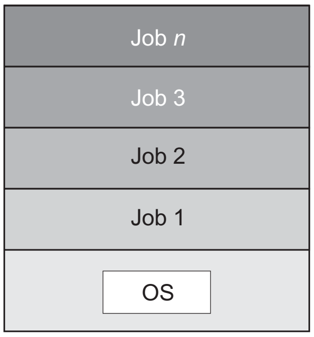


Fig. 1.7 Memory structure consisting of multiple jobs

The process for spooling was very simple. First, the input from input device would be read to the disk. Whenever the CPU would be free, the OS loaded the job from the hard disk to the memory and CPU would execute the job. After executing the job, it would store the results again to the disk. Later on, the results would be sent to the output device (see Fig. 1.8). Thus, CPU was busy in computation of another job while first job was busy in waiting for I/O. In this way, *multi-programming* concept came into picture. The requirements for the multi-programming increased the size of the OS because now the OS had the components for spooling, memory management (memory partitioning and keeping the account of jobs in the memory slots), managing multiple jobs, scheduling of tasks to CPU, and many more. In fact, this was the point where all other components of the OS were added. In other words, multi-programming was the basic concept around which all concepts of modern OS have been developed.

Although multi-programmed systems were able to improve the performance in terms of CPU time and throughput, still the jobs were batched and, therefore, there was no interaction of the job and the programmer. In fact, batch systems were not suitable for every type of job. There are certain jobs that need attention of user, i.e., interactive jobs. Due to batch of jobs, there was a long gap before the programmer would get the result back. Moreover, if there was a single error somewhere, the whole cycle of batch processing would get repeated causing further delays in getting the output. As a result, the debugging process became a time-consuming and irritating process. It was felt that even first-generation computers were better as they at least had the machines for a fixed time slot. Thus, the major drawback of multi-programmed batch systems was lack of user--programmer interaction with their jobs.

The solution found for the interaction between the job and the user was to have the dedicated dumb terminals for each user connected to the main system having CPU whose time would be shared among the multiple users. (Students should note that at that time, we did not have personal computers.) The idea was that user should have a feeling that he/she is using the system alone. Instead of submitting the job in a batch, the user would directly submit the job to the system. In this way, the user would be able to interact with his job. But the basic concept, i.e., multi-programming was still there. The jobs being submitted to the system would be stored in the partitioned memory. As the CPU time on the main system was shared among multiple users, these types of systems were known as *time-sharing multi-user systems*.

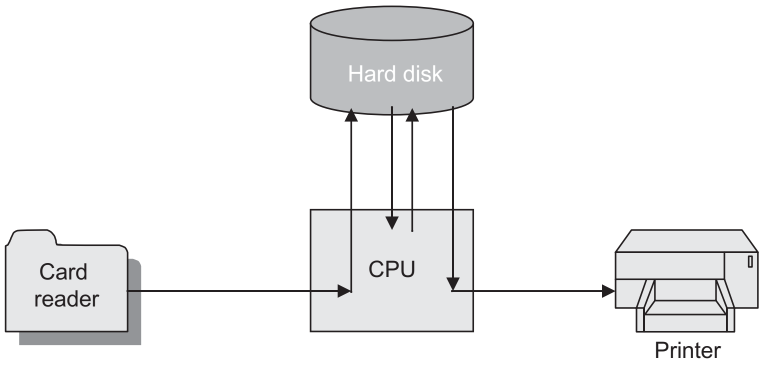


Fig. 1.8 Spooling

One of the earlier time-sharing systems was *compatible time-sharing system* (CTSS) developed at MIT. This system got popular as it supported many interactive users as compared to the batch systems. With the success of CTSS, it was thought that there should be a powerful machine which would support thousands of users at a time. For this type of project, MIT, Bell Labs, and General Electric came together with a system called *MULTiplexed Information and Computing Service* (MULTICS). But the idea of supporting thousands of users on Mainframe computers of that time did not work. Due to many reasons, the project could not be a success. Bell Labs dropped the project. Later on, General Electric also withdrew from the project and even left the computer business. But the story of time-sharing systems does not end here. MULTICS, despite the commercial failure, had a great impact on the future development of operating systems.

One of the scientists, Ken Thompson of Bell Labs, working on MULTICS thought of the idea of running the single-user version of MULTICS on PDP-7, a discarded minicomputer that time. With great surprise, Thompson's idea worked and it started supporting multiple users. Encouraged with his success, Brian Kernighan, Dennis Ritchie, and his other colleagues joined the project. Brian Kernighan started calling this project as UNICS (*UNiplexed Information and Computing Service*); however, later it was renamed as UNIX. After this, UNIX was ported to PDP-11/70, PDP-11/45, and PDP-11/70, the other modern computers of that time. These computers also had the mechanism of memory protection hardware. But the major problem realized at that time was that porting UNIX on different machines was becoming a tough job because UNIX was written in assembly language, and new hardware required rewriting of the code. It was surveyed and found that no language was suitable for making UNIX as a portable operating system. So it was realized that a new language should be designed for this purpose. Ritchie developed a language called 'C' for rewriting the UNIX. Finally, with the efforts of Thompson and Ritchie, UNIX was rewritten in 'C' high-level language and its portable version was prepared. In this way, the path of a time-sharing multi-user operating system was paved that dominated the world for a long time after that.

1.3.4 Fourth Generation

Undoubtedly, UNIX was a major achievement as a time-sharing multi-user OS in the third generation. Now the users do not need to wait for the response of their jobs for long hours. If there was an error in the user program, the user could debug it at that time only instead of waiting for a long time as in the batch systems. Thus, there was an interactive session between the user and the job. But, still this was a multi-user system which was sharing the processing time of a single system. It was desired that there should be a single system for a single user, which is not shared with others, i.e., the demand for personalization of processing time. However, this was not possible on the minicomputers of third generation.

With a leap of time, hardware technology further advanced and now it was possible to have thousands of transistors on a very small area of silicon chip due to LSI (Large Scale Integration) and VLSI (Very Large Scale Integration) technology. With the result of this success, the size of computers reduced beyond imagination. Due to this architecture, the processing speed of CPU further increased. This was the sunrise of microcomputers, later on called as *personal computers* (PCs). It was possible now to have a personal computer as compared to Mainframe or mini computers system shared by many users. Now the question was to design the OS for personal computers as the others present were not compatible with them. As Intel 8080 was the first microprocessor for personal computer, there was a need to design an OS for it. Gary Kildall in Intel designed an OS called '*CP/M*' (control program for microcomputers). With the success of CP/M, Kildall formed his own company Digital Research to support this OS for other microcomputers like Zilog Z80.

After some years, IBM came into the market with IBM PCs. IBM with Bill Gates hired Tim Paterson who had written one OS known as *disk operating system* (DOS). Tim modified DOS according to the modifications desired by IBM and came up with Microsoft DOS (MS-DOS). This OS with PCs revolutionized computing and became very popular among general users. On one hand, Intel was coming out with many new advancements in microprocessors like 8086, 80286, 80386, and 80486, and on the other, Microsoft modified MS-DOS and released many versions as per the microprocessors released. A number of utilities like dBASE, LOTUS, Wordstar and languages like BASIC, COBOL, C under DOS were also developed. This made programming and other jobs on the personal computer a very convenient task for a general user, which was not possible till third generation.

Despite the success of MS-DOS on PC, UNIX was also being modified with many versions to cater to many features of operating systems. The major and unique success point of UNIX was still multi-user time-sharing system which no other operating system of that time provided. Even DOS versions were also being updated as an impression of UNIX, e.g., the hierarchical file system incorporated in DOS was based on UNIX file system only. Microsoft was aware of the success of UNIX due to multi-user capability. Intel 80286 and other 80x86 family microprocessors were very fast in speed and were able to execute the jobs of multiple users. But MS-DOS on PC was designed only to handle a single-user job. So having the impression of multi-user feature of UNIX on mind, Microsoft came up with XENIX. After this, IBM also joined Microsoft for developing an OS known as *OS/2* for multi-user feature to be installed on 80286- and 80386-based systems.

The CP/M and MS-DOS operating systems were based on the commands to be typed by the user. It means whatever the task we wanted to do on these systems, we need to remember or refer the appropriate commands first, type them, and then perform the operation. This was becoming cumbersome to have so many commands to work with. Moreover, we needed to understand the hierarchical file system to work with the files, which was not user friendly. Thus, there was now another goal for OS developers—user friendliness and convenience. The research for this goal was being performed at Stanford Research Institute by Doung Engelbart who invented GUI (graphical user interface). This GUI concept was adopted by Steve Jobs who was working with Apple Computer at that time. Jobs's first attempt as *Lisa* system failed as it was too costly. His second attempt as *Apple Macintosh* was commercially successful not only due to its cheaper cost but also because of its user friendliness. The convenience and user friendliness concept relieved the users from command system and was very easy to learn.

Later on, Microsoft also realized the need to have an OS with GUI as impressed with the success of Apple Macintosh. By this time, Intel came up with 80386- and 80486-based systems which were faster as compared to the previous ones. This speed factor invented the graphic displays. Later on, Microsoft designed GUI-based operating system called *Windows*. But, Windows was just a graphical interface running on top of MS-DOS. It was in 1995 only that Windows 95 was released as a true operating system. After this, Windows has continued to rule over the world with many evolving versions: Windows 98, Windows NT, Windows NT 4.0, Windows 2000 (Windows NT 5.0), Windows Millennium, Windows XP, Windows Vista, and Windows 2007. The latest version is Windows 2008.

Intel continued to release the faster microprocessors (80486 and Pentium family), and applications of the users were becoming complex. At the same time, users demanded to execute multiple windows at a time in Windows OS. Every user wanted to open many tasks at a time on the system. This motivated to have an OS which would handle many tasks of a single user. This was termed as *multi-tasking* and implemented in many versions of Windows mentioned earlier.

The success of Windows as a user-friendly OS influenced the UNIX developers as it was lacking GUI. This motivated them to incorporate GUI features and come up with *X Windows*. X Windows included only basic window management. Another version having complete GUI features was released, known as *Motif*.

Another advancement in the hardware technology was the network system. In a network system, there were some basic functionalities like file transferring, remote login, etc. To provide these functionalities to a general user, network interface controller and a low-level software were required. Therefore, another type of OS was designed, known as *network operating system*. Similarly, distributed systems were developed, and thus *distributed operating systems*. The distributed systems were also network systems but there was a difference that in network systems the users were aware of the network and computer in that network. For example, if a user is copying a file to another computer through network, then user must have the address of that destination computer. It means the user has the knowledge of the task and is aware of the location in the network where the task will be performed. On the other hand, in a distributed system, the user submits the complex task and gets the result back. He does not know how the task has been divided among multiple tasks; at which nodes in the network these tasks have been sent for processing. All these functionalities are performed by a distributed operating system without the knowledge of the user. Moreover, a distributed system requires a different operation compared to a centralized system such as distributed scheduling, i.e., scheduling of various tasks on various processors, distributed synchronization among tasks, distributed file systems, etc.

The evolution of all major OSs is given in Table 1.2.

Table 1.2 Evolution of different operating systems

Generation	Period	Computer architecture	Problems and development of OSs
First	1940s–1950s	Vacuum tubes based technology, plug boards and punched cards, magnetic core memories	No operating system
Second	1950s–1960s	Transistors based technology, Mainframe computers, line printers, magnetic tapes, assemblers, linkers, loaders, compilers, FORTRAN, COBOL	Set-up delay problem due to loading and unloading of tapes in earlier computer systems. CPU was idle. Jobs of users prepared with same programming language were batched together. Automated job sequencing Resident monitor Batch systems Mismatch between the speed of CPU and I/O devices Offline operation with magnetic tapes Tapes were sequential access devices

(*Contd*)

(*Table 1.2 Contd*)

Third	1960s–1980s	IC-based technology, Minicomputer Magnetic disk	Hard disks came into existence Spooling Multi-programming Multi-programmed batch systems Lack of user–programmer interaction with their jobs in multi-programmed batch systems Time-sharing multi-user systems CTSS MULTICS UNICS UNIX UNIX written in C
Fourth	1980s–Present	LSI- and VLSI-based technology, Microcomputer	CP/M for PCs MS-DOS Multi user facilities were not there in DOS XENIX OS/2 No user friendliness and convenience due to command driven and complex file systems Apple Macintosh Windows Multi-tasking Multi-threading X Windows Motif Network operating systems Distributed operating systems

1.4 TYPES OF OPERATING SYSTEMS

We have traced the history of OSs and seen how they have been developed. Today, a variety of OSs exist according to the project environment and requirements of the user. For daily use, we use Windows OS as it has become a general-purpose OS today. On the other hand, if the project is embedded with a real-time system, then Windows will not suffice. Instead, a real-time OS (RTOS) will be required to meet the requirements of real-time system project. If we are working on a multiprocessing distributed system, then the OS should be able to distribute the processes on various processors, coordinate between distributed processes, etc. Thus, OSs are there to meet our needs. We should identify the requirements and project types and then select an appropriate operating system. Here we describe various types of OSs.

1.4.1 Batch Processing Systems

The batch processing system was developed as a result of more set-up time for execution of different types of user programs. But today, we do not have the problems of set-up time. However, batch processing can be used for the user jobs which do not want user attention. These jobs can be combined in a batch and sent for execution without the intervention of the user. Thus, batch processing systems take a sequence of jobs in a batch and executes them one by one without any intervention of the user. The main advantage of batch processing is to increase the CPU utilization. The batch processing, obviously, is not meant for a quick response to the users, but it is still used to quantify the user service turnaround time. The turnaround time of a user job is the time since the job was submitted to the system to the time when the user gets the result back.

1.4.2 Multi-programming Systems

Multi-programming is a very basic concept today. In evolution of operating systems, it was described that multi-programming means to place several programs or jobs in the main memory instead of a single program. It means that now several jobs are ready to be executed, but CPU can execute only one job at a time. Then how do we execute all the jobs in the main memory? The idea is switching between the jobs.

There can be two types of instructions in a program: CPU bound and I/O bound. CPU bound instruction means when CPU has an instruction for processing or computation. I/O bound instruction means there is a request to an input or output device to read or write. It means during the I/O bound instructions, CPU is not doing work and is idle, i.e., the job which CPU was executing is now waiting for an I/O service. It has been observed that most of the time in a job is wasted in waiting for I/O. When there is a single program in memory or in monoprogramming concept, the CPU sits idle and waits for I/O to complete and then moves to next CPU bound instruction. Since, due to multi-programming concept there are many jobs ready in the main memory, the CPU can switch to its second job while the first is waiting for an I/O. If the second job also reaches an I/O bound instruction, then CPU switches to another job and so on. The CPU comes back to its previous jobs and executes them in the same fashion. With this concept, the CPU always has a job to execute and remains busy. For example, see Fig. 1.9; there are two programs

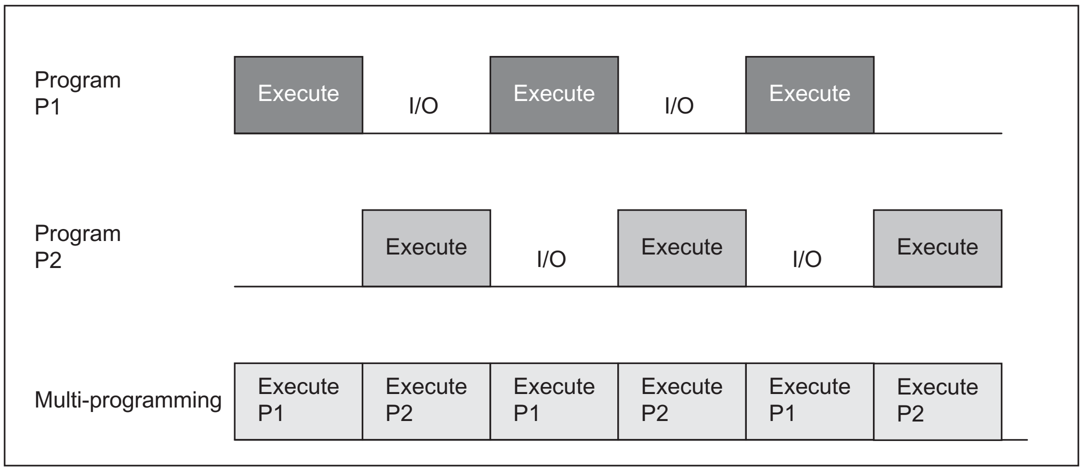


Fig. 1.9 Multi-programming

P1 and P2. With monoprogramming, there would be only P1 in the main memory and P2 will arrive in the memory and be executed only after the execution of P1. But with multi-programming, both programs will be stored in the memory. Suppose if P1 is first sent to CPU for execution, then after some time if P1 waits for some I/O, then in monoprogramming, CPU will sit idle. But in case of multi-programming, it will switch to P2 and start executing it. With this arrangement, the CPU does not sit idle; and in this example, both P1 and P2 will be executed in the same time as required to execute P1 in monoprogramming. Obviously, there is higher CPU utilization (ideally in this example, it is 100%) and throughput is doubled. Although this is an example of an ideal case, there is high CPU utilization and throughput in multi-programming systems.

The major benefits of multi-programming systems are as follows:

- *Less execution time* As compared to batch systems, the overall execution time for users' job is less because of spooling and switching between the jobs frequently by CPU.
- *Increased utilization of memory* Instead of storing a single program in the memory as done before, now more than one program is stored, thereby utilizing the main memory.
- *Increased throughput* Since in multi-programming, CPU does not sit idle and switches between all the jobs, the number of jobs executed in time *t* is more as compared to batch systems, thereby increasing the throughput. Throughput may be given as follows.

Throughput = Number of jobs completed per unit time

Throughput is increased if *degree of multi-programming* is increased. Degree of multi-programming is the number of programs in main memory. As we increase the number of programs in memory, i.e., increase the degree of multi-programming, throughput also increases. However, this increase in throughput depends on two factors. First, how much memory for storing programs is available and second, the type of program. If the programs under multiprogrammed execution are either CPU bound or I/O bound, the throughput will be low. If there is a proper mix of these two types of programs, only then the throughput will be improved.

Example 1.1

There are three jobs running in a multi-programming system with the following requirements:

Job 1: Requires disk after every 2 min (device service time including wait and access = 2 min). Total processing time = 6 min.

Job 2: Requires printer after every 5 min (device service time including wait and access = 2 min). Total processing time = 7 min.

Job 3: Requires disk after every 3 min (device service time including wait and access = 2 min). Total processing time = 5 min.

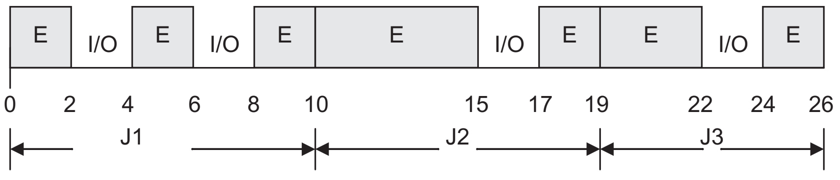
Prepare a timing chart showing the CPU and I/O activities of the jobs. Compute the total time for execution using mono-programming and multi-programming and then compare the results.

Solution

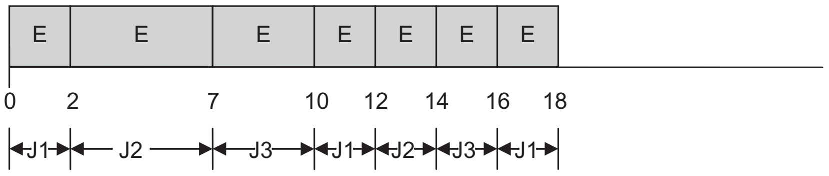
We represent our jobs with J1, J2, and J3 and the execution of the jobs as E. First see the timing [Fig. 1.10(a)] with monoprogramming and then [Fig. 1.10(b)] with multi-programming. From these two diagrams, it is clear that about 30% of time has been saved with multi-programming and CPU was busy instead of being idle just for waiting the I/O to be completed.

1.4.3 Multi-user Time-sharing Systems

Batch systems and multi-programmed batch systems do not provide immediate response to the user. If one user submits his/her job, he/she has to wait for the execution of all the jobs in that batch and then get the output. In this way, waiting time of a user is more and he/she is not



(a) For monoprogramming



(b) For multi-programming

Fig. 1.10 Timing diagram

in direct touch with his/her job. If something goes wrong in the job, the user needs to correct it, submit it in another batch, and again wait for a long time to execute the full batch. This behaviour of batch systems and multiprogrammed batch systems were due to non-interactive input and output devices. With the invention of interactive devices like keyboard and video terminals, another paradigm was designed wherein multiple users with their terminals (having no processors) were connected to a computer system (with processor) to perform their jobs. In this arrangement, the jobs of multiple interactive users were placed in the main memory instead of batched jobs. It means that multi-programming was still used here. This system was called *multiuser* as it supported multiple interactive users. It was also known as *time sharing* as CPU time of main computer system was shared among multiple users to execute their jobs. Thus, multiuser time-sharing systems are the systems where multiple interactive users connected through their dumb terminals (for interface only) access the main computer system (with CPU) to perform their jobs (see Fig. 1.11).

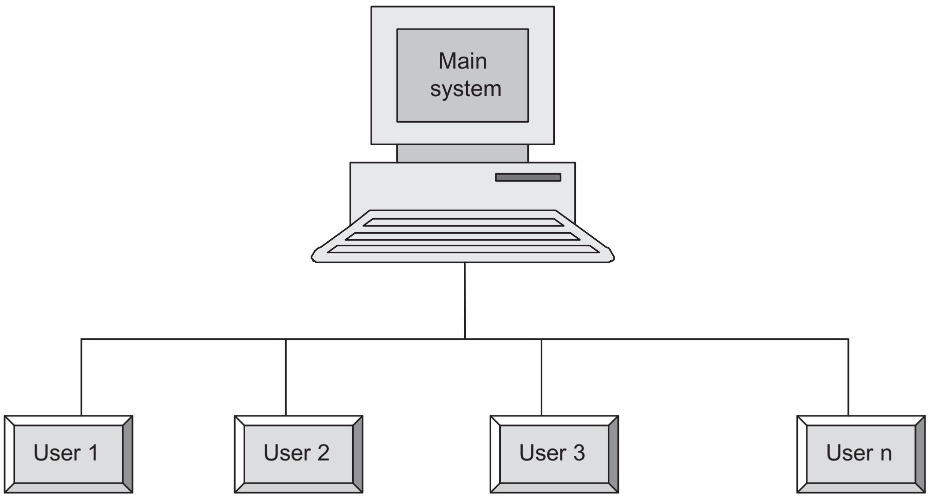


Fig. 1.11 Multi-user time-sharing system

As mentioned above, this technique uses multi-programming, and CPU time of main system is being shared among multiple users. However, due to the processor's speed, each user has the impression that the system is dedicated to him/her only. Therefore, CPU switches from one job to another job on regular intervals to have a fair distribution of its time for the users. For this fair distribution, a *time slice* for each user may be fixed. By using this time slice, the users' jobs are scheduled in such a manner that each job gets equal chance for computation. In this way, the major benefit of time-sharing systems over multiprogrammed batch systems is the improved response time. The response time is the time between submission of a job to the system and its first reaction/response to the user. The better response time, in turn, improved the productivity of the user as he/she gets the quick response to his every job.

The major benefits of multiuser time-sharing systems are as follows:

- *Multiuser facility* With time-sharing paradigm, it is possible to connect multiple users to a system where each user presents the job to the system and gets the response.
- *Improved response time* The major benefit is the *response time* for a user's job which was not possible in batch systems. The user now is in direct touch of his job and due to easy interface, he views everything regarding his job.
- *Improved debugging and productivity* From the programmer's point of view, debugging of the programs is now easy because the user can easily view his mistakes. In this way, he quickly modifies the program and runs again, thereby increasing the productivity also.

1.4.4 Multi-tasking Systems

In most of the literature, multi-programming and multi-tasking have been used interchangeably. In the literary sense of both these terms, they seem to mean the same thing and one may get confused with these two terms. Similarly, time-sharing and multi-tasking are also used interchangeably. Let us understand these terms first.

As mentioned above, multi-programming was the basic concept wherein more than one program were stored in the main memory and first batch systems were developed around this concept of multi-programming. Due to no interaction of users with their jobs in batch multi-programming, multiuser time-sharing systems were developed. So it should be noted here that in history, the term time-sharing was used for multiuser systems. Today time-sharing has been used as a scheduling technique which should not be confused with any term like multiuser or multi-tasking. After this, Windows was developed for personal computers such that a single user working on a PC can open multiple windows. The user can open the web browser and at the same time, open a Word file to edit (see Fig. 1.12). In this way, the user is able to open many windows or tasks and work on them. This is known as multi-tasking where a single user works on multiple tasks on the PC. With the availability of high speed of processor, the user has the illusion of working in parallel on multiple tasks. But it is the time-sharing scheduling technique which has made it possible. And we know that time-sharing technique was also used in multiuser systems. This is the reason that multi-tasking and time-sharing are mixed up. But it should be made clear that multiuser and multi-tasking are different terms, and time-sharing is the scheduling technique in both of them. Moreover, the multi-programming technique (more than one program/task in the main memory) is inherent in both multiuser and multi-tasking systems.

To make it more clear to the readers, let us summarize some definitions:

- *Multi-programming* Place more than one job/program/task in the main memory.

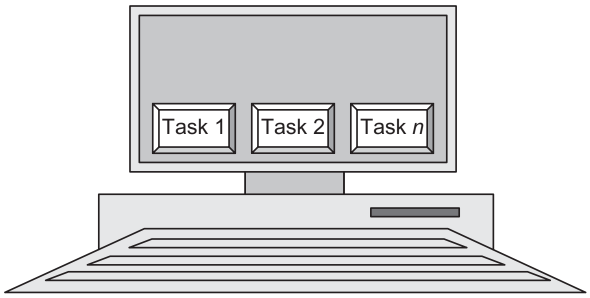


Fig. 1.12 Multi-tasking System

- *Multiprogrammed batch systems* Place more than one job/program/task in the main memory of a batch prepared for same type of jobs, and execute them by switching between them.
- *Multiuser systems* Place more than one job/program/task in the main memory of the main computer system. Here jobs come from the different users who are connected through terminals to the main computer. The jobs are scheduled by time-sharing technique.
- *Multi-tasking systems* Place more than one job/program/task in the main memory of the system. The jobs here are of a single user working on the system. The jobs are scheduled by time-sharing technique.

1.4.5 Network Operating Systems

The network operating system is the earliest form of operating system that coordinates the activities on a network system. Network operating system may be considered as loosely coupled operating system software on a loosely coupled hardware that allows nodes and users of a distributed system to be quite independent of one another but interacts in a limited degree. In a network, each node has its own local operating system. A user sitting on a node may work as on the local machine through its local operating system. However, on the network system, there may be some control operations that may be performed by the user sitting on his/her machine. In other words, the user working on a node is also able to perform non-local functions. For example, the user may remotely log on to some other node. Similarly, the user may transfer the files to another node also. For the functioning of these non-local operations, the operating system software is required to coordinate the activities. Here the role of network operating system starts. The network operating system may be considered as another layer of software on the operating system on a local machine. This layer works between the user computations and the kernel on the local machine. The processes of user first contact the network operating system. If the operation to be performed is local on the node, the network operating system passes the request to the local operating system on the node. But if the operation to be performed is non-local, the network operating system contacts the network operating system on the node.

A network operating system also targets the resource sharing across multiple nodes of the network where each node has its own local operating system and a layer of network operating system. Each node on the network is able to perform some control operations that are run locally as well as on some other node on the network. However, to work on a network system using network operating system, the user must be aware of the network nodes and their access rights to perform the control functions. For instance, if a user wants to log on to some other node on the network, i.e., remote login, he must know the address of the node and must have permission to log on the system. Similarly, while transferring the files, the user must explicitly transfer the file from his machine to another one, i.e., he must be aware of where all files are located and where they will be copied. Thus, in network operating system based system, a user must know where a resource is located in order to use it leading to poor transparency of system resources and services.

1.4.6 Multi-processor Operating Systems

In the technological evolution of computer systems, there was a desire for parallel processing with the help of more than one processor instead of only one. This has been realized through multi-processor systems. Multiprocessing systems contain more than one processor and share other resources. These types of systems are very useful for engineering and scientific applications by processing data in parallel on multiple processors. Another category of application suitable in multiprocessing environment is mission-critical and real-time systems. Since these systems are specially designed for defence systems, it is expected that they will continually work in warfare conditions. Therefore, in these types of systems, besides parallel computation, there is a high demand of fault tolerance and graceful degradation of services when any of the processor fails. Thus, multiprocessing systems are most suitable for these applications. Multiprocessing systems offer the advantage of increased throughput due to multiple processors, are economical to work on, and have increased reliability due to fault tolerance.

It is obvious that for multiprocessing systems, different operating systems are required to cater to the special requirements. These are called multiprocessing operating systems. These operating systems have more challenges as compared to single-processor systems. Since in this environment there are multiple processors, all of them should be busy. The processes should be distributed on various processors for parallel computation. The process scheduling is another challenge as it is needed to schedule multiple processes on multiple processors. Moreover, the coordination of various processes should also be taken care of. Different inter-process communication and synchronization techniques are required. In multiprocessing systems, all processors share a memory; therefore, there is a need to check that all processors operate on consistent copies of data stored in shared memory.

1.4.7 Distributed Operating Systems

Distributed systems are also multi-processor systems but with the following differences:

- Distributed system works in a wide area network.
- Each node in a distributed system is a complete computer having full set of peripherals including operating system.
- The users of a distributed system have an impression that they are working on a single machine.

Resource sharing is the main motive behind distributed systems. If we want to take advantage of hundreds of processors, it may not be possible to have all of them on a single board. But the multiple processors are realized as a single powerful machine in a network system and this machine is known as a distributed system. In this way, a number of users can share the resources of all the machines on the distributed system.

Besides the resource sharing, the distributed systems also provide computational speed up by partitioning a computation into some subcomputations which are distributed and run concurrently on various nodes on the system. A distributed system also provides the enhanced availability of the resources through redundancy and communication paths thereby increasing the reliability of the system. For example, a distributed file system places files on separate machines and allows many users to access the same set of files reliably providing the view of a single file system.

Distributed operating systems have been designed for this special type of system. These operating systems providing distributed computing facility employ almost same communication methods and protocols as in network operating systems. But the communication is transparent to the users such that they are unaware of the separate computers that are providing the service. The following are some important tasks to be met by distributed operating system:

- Since distributed systems need to access any resource or transfer any task on any node, there are three types of migration provided by the operating systems:

  Data migration Transferring the data from one site to another site
  Computation migration Transferring the computation on a particular node
  Process migration The process or its subprocesses may also need to be transferred to some other nodes due to some reasons like load balancing, computation speed, etc.

- Distributed OS must provide the means for inter-process communication. Some of the methods are as follows:

Remote Procedure Call A process on one node may invoke a function or procedure in a process executing on another node.

Remote Method Invocation Allows a Java process to invoke a method of an object on a remote machine.

CORBA (*Common Object Request Broker Architecture*) It is a standardized language that supports different programming languages and different operating systems for distributed communication.

DCOM (*Distributed Component Object Model*) Another standard developed by Microsoft included in Windows operating system.

- Due to multiple processes, synchronization methods should be supported by the operating system.
- There may be deadlock when processes distributed over several nodes in a network wait for the resources not released by other processes. Therefore, deadlock should also be handled by OS.

1.4.8 Real-time Operating Systems

In time-sharing systems, there was a drawback, i.e., if there was more load on the system, the response time was more and further increased with the increase in load. But there are some computations in a system which cannot bear the delay. Therefore, after the development of time-sharing systems, new kinds of computer systems were developed in 1980s. In these systems, response to a user request has to be immediate or within a fixed time frame, otherwise the application will fail. This is known as *real-time processing*. This type of processing is largely useful in defence applications which are mission specific, i.e., if there is no timely response, there might be loss of equipment and even life. Therefore in these systems, there are deadlines of time which should be met to prevent failures, otherwise the purpose of the system is lost. For example, suppose there is an electric motor being controlled through a computer system. The motor running at a speed, if crosses a threshold speed, will burn. The system is controlling the motor in such a way that if motor crosses the threshold, it will lower the speed of the motor. Now, when motor is crossing the threshold speed, and system does not respond in that time period, the motor will burn. So, this is an example of real-time system which in case of failure results in loss of equipment. Similarly, there are many defence applications like guided missile systems, air traffic control systems, etc. which in case of failure may result in loss of life.

Real-time systems are of two types: *hard real-time* systems and *soft real-time* systems. The systems that have hard deadlines and must be met are called hard real-time systems. All defence applications are of this type. There is another type known as soft real-time system where missing of some deadline is acceptable. For example, in a video conferencing system, if some audio or video data are somehow delayed for a fraction of time, then it may be acceptable and there is no harm. Thus, digital audio, multimedia systems, virtual reality are all examples of soft real-time systems. However, missing the deadlines does not mean that they are not real-time systems. The delay of soft real-time systems must be bounded and predictable and should not be infinite.

Real-time operating systems (RTOS) are there to meet special needs of a real-time system. They have the major characteristic of providing timely response to the applications besides other facilities. The major challenge for an RTOS is to schedule the real time tasks. In a real-time system design, all deadlines requirements are gathered and analyzed. The RTOS schedules all tasks according to the deadline information and ensures that all deadlines are met. Another feature of a real time system is to have fault tolerance. Since a real time system must work continuously in every condition, therefore in case of any hardware or software failure, the system should not stop working. To achieve this target, fault tolerance is provided by means of redundancy both in hardware and software. For example if one processor fails, another processor in the standby will take over the charge and the system continues to work. The RTOS must use some special techniques such that the system can tolerate the faults and continue its operations. Obviously, with the fault tolerant feature, there is degradation in the performance of the system. But OS should take care that this degradation is graceful, i.e., no critical functioning should be stopped or delayed.

1.4.9 Embedded Operating Systems

Embedded systems are specialized systems that tend to have very specific tasks. The last decade has filled our daily life with embedded systems. From the household systems to defence systems, the embedded systems are seen everywhere. Either you purchase the toys for your children or the smartphone for yourself, these systems have dominated every walk of life. Washing machines, televisions, and cars are other examples where these systems are being used.

Embedded systems have also operating systems but they are not generalized ones. The user uses these devices without any awareness of operating systems. Embedded operating systems are there to perform all the basic functionalities in these systems like initialization, task management, memory management, etc. but with little or no user interface. Thus, in the embedded systems, there are operating systems but not in the same structure as found in general purpose computer systems.

A large number of devices categorized as consumer electronics are mobile. They are better known as mobile devices or hand-held devices. One of the category of mobile devices is personal digital assistants (PDAs), such as palm top computer, are hand-held devices that combine elements of computing, telephone/fax, Internet, and networking in a single device. A typical PDA can function as a cellular phone, fax sender, web browser, and personal organizer. Thus, these devices allow us to access the email, messaging, web browsing, work on documents, and much more. The examples for PDAs are Palm Pilot, Toshiba Pocket PC. PalmOS is a well-known OS for them. The second category is mobile phones and smartphones. Smartphones combine both mobile phone and hand-held computers into a single device. They allow users to store information (e.g., e-mail), install programs, along with using a mobile phone in one device. The examples of operating systems for smartphones are: Symbian OS, iPhone OS, BlackBerry, Windows Phone, Linux, Palm WebOS, Android, and Materno.

Another category of mobile devices is smart cards. Smart cards have the capacity to retain and protect critical information stored in electronic form. The smart card has a microprocessor or memory chip embedded in it. The chip stores electronic data and programs that are protected by advanced security features. When coupled with a reader, the smart card has the processing power to serve many different applications. Smart cards provide data portability, security, and convenience. Smart cards are being used in various sectors, such as telephone, transportation, banking, healthcare transactions, etc. There are two basic types of smart cards: contact and contact-less smart cards. Contact cards have a 1-cm-diameter gold-plated pad that has eight contacts on it. These contacts are in turn wired to a microchip underneath the pad. The microchip can be a memory-only chip or a microprocessor chip containing memory and a CPU. Memory cards are used mostly as telephone cards, whereas microprocessor cards can be used for multiple applications on the same card. Although both cards can have stored value and stored data areas, the microprocessor card can in addition process the data since it contains a CPU, RAM, and an operating system in read only memory (ROM). Contact-less cards not only have an embedded microprocessor chip, but also contain a miniature radio transceiver and antenna. They only operate within close proximity to the reader. Instead of inserting the card, we simply pass the card close to the reader. Contact-less cards tend to be more costly than contact cards and are best suited for transportation and building access applications

The smart card's chip operating system (COS) is a sequence of instructions permanently embedded in the ROM of the smart card. The baseline functions of the COS which are common across all smart card products include

- management of interchanges between the card and the outside world, primarily in terms of the interchange protocol
- management of the files and data held in the memory
- access control to information and functions (select file, read, write, and update data)
- management of card security and the cryptographic algorithm procedures
- maintaining reliability, particularly in terms of data consistency, and recovering from an error

There are some challenges for the designers of the operating systems for mobile devices. Some of them are as follows:

- All the mobile devices have a very small memory. So the memory must be managed efficiently.
- All the devices have a slow power CPU as faster CPU will require more power and thereby a larger battery. And larger battery cannot be there in a small mobile device. Therefore, the operating system should not load the processor with heavy computations.
- Devices like mobile phones and smartphones have a small screen area. So the contents should be mapped to the available size of the display screen.

The features of major types of operating systems discussed above are given in Table 1.3.

Table 1.3 Types of operating system

Type of operating system	Features/benefits	Example	Applicable to which type of application
Batch systems	More than one job can be stored in main memory	FMS (FORTRAN monitor system), IBM's operating system for 7094	Background jobs in which the user interaction is not necessary

(*Contd*)

(*Table 1.3 Contd*)

	Batches of same type of jobs can be executed quickly		
Multiuser systems	Jobs of different users who are connected to a main computer are executed through the multi-programming Interaction of jobs with the user is possible Debugging is easy	CTSS by MIT, TSS by IBM, MULTICS, UNIX	When multiple users need to share a single system
Multi-tasking systems	Multiple tasks of a single user can be opened on the system through multi-programming	Windows	When a user wants to open and work simultaneously on many windows on the system
Network systems	The user is able to connect to another machine and perform many operations The user is aware of the location of the network node where he/she wants to connect	Novell Netware, Windows NT, Windows 2000, Windows XP, Sun Solaris	When a user wants to remote log on to a system, wants to transfer a file, etc. on a network system
Distributed systems	When multiple nodes of a wide network realized as a powerful machine sharing the resources on the network. The users are not aware where their processes are being sent and executed.	Amoeba, V system, Chorus	When computational speed and resource sharing is required and implemented through various full computer systems in a network
Real-time systems	Used to handle time-bound responses to the applications	pSOS, VxWorks, RTLinux, etc.	Applicable to systems which require time-bound response, i.e., for the real-time processing systems
Embedded systems	Specialized systems with size, memory and power restrictions	Palm Pilot, Toshiba Pocket PC, Palm OS, Symbian OS, iPhone OS, RIM's BlackBerry, Windows Phone, Linux, Palm WebOS, Android and Maemo.	Used in consumer electronics items, mobile phones, smart cards, etc.

1.5 GOALS OF AN OPERATING SYSTEM

We have seen how operating systems have been developed in the past. It can be noticed that as soon as the concept of multi-programming came into picture, a number of problems started to occur. The solution of these problems in fact contributed for further development of operating systems. All these operating systems have been developed keeping in view the convenience of the user of the system. As discussed, there was a time when all the tasks related to programming had to be done by the user only, whether it is loading the program in memory or debugging the program. The journey from multi-programming through multiuser is still continuing with multi-tasking and multi-threading. The central theme is the user's convenience. Either it is the throughput, i.e. the number of tasks being performed, or the response time of a user's job, or the user wants to execute more than one tasks at a time or the environment wherein the user can perform the tasks conveniently without worrying about the CPU time, memory partitions, memory size, etc. All these developments signify that an operating system provides an environment to a user such that he concentrates on his job instead of being concerned about the hardware.

Another point that we have observed through the development of operating systems is that operating systems have been designed to utilize the hardware and other resources. For instance, CPU time had always been the target for its utilization. From the batched multi-programming systems to multi-tasking systems, CPU time had been the central resource to be utilized. The goal of CPU usage also changed as the generation of computer systems changed. In initial operating systems, the goal was the throughput and that CPU should be made busy in executing more number of jobs. But in the recent operating systems, the goal has changed to response time, i.e., CPU executes the jobs to provide a user the minimum response time for his jobs in multi-tasking environment. Another important resource is memory. With the invention of multi-programming, many jobs started residing in the memory. Therefore, it was necessary to keep maximum jobs in the main memory and utilize it as much as possible. Even the concept of virtual memory was given with the help of which the user does not need to worry about the size of the memory to write a program. In fact, it is possible to have a program larger than the actual size of memory. There is another concern of utilization—devices utilization. All devices should be shared and utilized fairly among the tasks. As we will go through the chapters in this book, we will realize the need of utilization of all the resources being used in the computer system. Therefore, to meet the needs of resources utilization, an operating system is required. Thus, the operating system provides an abstraction to the user of a system that he is best utilizing the resources of the system even without his knowledge.

As a result of resource utilization, another goal for operating system has been added, i.e., resource allocation. In multi-programming environment, there is need to allocate resources to various tasks. For instance, when there are many tasks in the memory, it is required that one of them will go to the CPU for execution. Someone has to take this decision for process allocation. The operating system has been designated to perform this function. It uses various scheduling algorithms according to the need and chooses one of the tasks in memory and sends it to CPU for execution. For memory utilization, there is a problem of memory allocation to a task, i.e., which vacant space should be allocated to a task? Similarly, devices are very limited as compared to the tasks. Therefore, again the operating system allocates the devices to the tasks in best possible manner. Thus, another goal in designing the operating system is to allocate resources to various tasks. Therefore, it is also known as *resource allocator*.

It should be noted that there is a side effect of multi-programming and multi-tasking. We know that more than one job resides in the memory in these concepts. Moreover, operating system is also loaded in the memory in a separate partition. However, it may be possible that one user accesses the memory region of another user or even the operating system. Till the development of DOS, there was no protection among users' jobs and operating systems. Therefore, the next goal for an operating system is to have protection in this form such that no user is able to access others memory regions and of operating system.

Let us summarize these goals as follows:

Convenience

The convenience of a user, who performs a job using the computer system, is the prime goal of an operating system. Let us understand the user requirements. Some of them are as follows:

Hardware abstraction/virtual machine

The user does not want to care for hardware resources and to access the devices because the details of all the hardware are too complex to work. The details of hardware also hinder the thinking process of a programmer. Therefore, operating system provides an abstraction layer between the user and the hardware so that the user can work on the hardware without actually knowing it.

Convenient programming environment

The process of program execution includes several steps. We should have a good editor to write a program, debugger to debug the program, linker, loader, etc. The operating system provides all these facilities to a programmer so that a convenient environment is present and the user can concentrate on the programming and nothing else.

Response time

As we know through the evolution of operating systems that there was a time when there was a gap of hours or even a day between the user's job submission and its first response of execution. With the invention of high-speed processors, the user desired to have immediate response from the job. This desire has resulted in the development of multiuser and multi-tasking operating systems.

Easy-to-use interface

The users' convenience has taken another shape when the use of GUI became popular. This was another goal for an operating system which resulted in Mac OS and Windows OS. The use of an operating system in the form of GUI makes a user understand the operation he wants to perform. Moreover, the user is relieved from remembering the commands which needed to be typed in older UNIX and DOS systems.

Resource Utilization/Management

We know that multi-programming introduced many new challenges for an operating system. Due to multi-programming, there was a need to partition the memory and allocate memory to various tasks. So memory partitioning and memory allocation jobs added to the operating system. As the number of users and tasks increased, there was need to utilize the memory efficiently. Thus, memory management techniques were devised.

Another concern that arose due to multi-programming was CPU utilization. When there was a single program to execute, there was no issue of memory management and CPU utilization. But when there are multiple tasks in memory, all compete for execution. Since CPU can execute a single task at a time, others need to wait. So the issue is to schedule the tasks such that CPU does not sit idle and all the tasks get the CPU time fairly.

Device utilization is another important issue for operating system. As the devices are less in number as compared to the number of users and tasks, there is need of controlled allocation to them. For example, if we allow all the users to access a shared printer, then there will be a chaos. It means we must have some mechanisms to share the devices. This device management is also supported by the operating system.

Likewise, there are many resources today in the computer system which need to be allocated and managed in a controlled manner. In this way, operating system's jobs to manage and utilize the resources are as follows:

i) grants access to resource requests from different users
ii) keeps track of which task is using which resource
iii) keeps account of usage of the resources
iv) resolves the conflicting requests from users
v) utilizes the hardware and other resources in best possible manner

Thus, it can be said that operating system acts as a resource allocator and resource manager in order to provide an efficient operation on computer system to the user. The efficiency is the major goal for an operating system which takes care of the best utilization and allocation of resources on the computer system.

Protection

Due to multi-programming and multi-tasking, there was a challenge that one user should not be able to access other user area in memory. Similarly, one user should not be able to access the operating system area in memory. For this purpose, hardware was first modified and then protection feature was added in the operating system (we will discuss this issue later in detail). Thus, an operating system should be able to protect the task of one user from another and operating system from any user.

1.6 FUNCTIONS OF OPERATING SYSTEM

We have understood what an operating system is and broadly what it does. Now it is time to discuss the various functions being performed by an operating system. The functions can be categorized as per two viewpoints: user view and system view. The user view is a top-down view of functions performed by an operating system, whereas system view is bottom-up view of functions performed by an operating system.

1.6.1 User View

The user view is to execute the user's task on the computer system. But a user does not want to be overwhelmed with the complex hardware details of the system. He simply wants an interface between his application and the hardware. There are many system programs or utilities to help him in developing his application. The operating system is also a system program developed to act as an interface between the application and the hardware. He is not concerned how the application will get resources from the system and get executed. All these jobs will be done by the operating system. Thus, an operating system does hardware abstraction for the user by hiding the complex details of the hardware, thereby providing a flexible user-friendly interface to the user for developing his application. In other words, operating system acts as a mediator between the application and the computer system that makes easy use of hardware and other resources without even knowing. From the user's point of view, the following are some functions performed by an operating system:

User Interface

The operating system provides the interface to use the computer system. There are two types of interfaces: command-driven interface and graphical user interface (GUI). As discussed earlier, in older systems, there was only command-driven interface. But with the invention of Windows, now almost every operating system provides GUI.

Program Development and Execution

For executing a program, there are certain tasks like loading the program in main memory, initializing and accessing I/O devices and files, scheduling various resources, etc. All these program executions are performed by the operating system without the knowledge of the user. Moreover, operating system provides some utilities such as editors, debuggers, etc., which although are not part of the operating system but are packed with the operating system for the convenience of the programmer.

Accessing I/O Operations

If you have written some programs in high-level language, then you write some standard input-output instructions according to the language being used. For example, in 'C' language, for reading the input the instruction is *scanf* and for output the instruction is *printf*. You do not care for the type of input/output devices and use only standard instructions for any type of devices. The operating system relieves the user from details of input/output devices and accesses them on behalf of the user.

Accessing File Systems

A file is a logical concept to store the user's data or program. A user creates the file using some editor and saves and retrieves the files conveniently through the operating system's interface. But the file as a logical entity is mapped to some physical memory. Operating system in background keeps track of memory provided to the files and performs all the operations related to file management.

Error Detection

While working on a computer system, one may encounter different types of errors. Either it is a hardware error or error in some user program. There may be a memory access failure or a device failure. A user may give a command for printing a file but there may be no paper in printer or there is a paper jam. Or the errors may be in user programs such as arithmetic overflow, divide by zero error, or accessing an illegal memory location. All these errors must be identified by the operating system and an appropriate action must be taken and the user should be notified through a message on the screen.

1.6.2 System View

Beyond the user's convenience, most of the functionalities are performed in background by the operating system. These activities are to manage or utilize the hardware and other resources of the computer system. Therefore, from the computer system's point of view, the operating system is a program that controls the allocation/execution of all the resources in the system. There are three major functionalities done by the operating system:

Resource Manager

The operating system is a program that controls the allocation/execution of all the resources in the system. In this way, operating system schedules and manages the allocation of all resources in the computer system. It is best called as a resource allocator and resource manager. If there are multiple processes, then their creation, deletion, and other operations are handled by the operating system only. Memory is to be provided to all these processes. It is the job of operating system to look for the available memory and allocate to the process. When multiple processes are simultaneously running there may be problems such as how to communicate with other processes and how to access the shared objects. Therefore, an operating system needs to also handle inter-process communication and synchronization. It utilizes the resources in the best possible manner. Users work on the computer using various files. The user sees these files as only a logical concept. The operating system implements the file systems in the physical memory. Similarly, I/O devices, which are not directly accessible to the user, are given access by the operating system. Thus, from the system's viewpoint, process management, file management, I/O management, memory management, etc., are all the functions performed by the operating system.

Control Program

The operating system acts as a control program in the sense that it protects one user's program from another. It is necessary in multi-programming because a user may try to enter other user's memory and even in operating systems' region. Also, it does not allow the users to access any I/O devices directly as the user may misuse them. It detects any exception in the system if it happens and informs the user. Thus, operating system acts as a control program that controls the user activities, I/O access, and all other activities performed by the system.

Virtual Machine Manager

A very different view to see the operating system is as a virtual machine manager. As we know that operating system acts as an abstraction which hides the complex details of the hardware from the user and presents a user-friendly environment to him. Even a programmer does not want to interact directly with the I/O devices. Interacting with I/O devices, memory, and hard disk is too clumsy and no one can easily work on these hardware details. Due to this fact, operating system provides a layer on the actual hardware on which it performs the tasks of the user. And to the user, it seems that all the work done is by the hardware. In other words, there is an illusion created by the operating system that there is a virtual machine that is performing all the work. Let us see how a virtual computer is created:

i) The operating system creates virtual processors by creating multiple processes.
ii) The operating system creates multiple address spaces out of the physical memory and allocates to various processes.
iii) The operating system implements a file system out of the hard disk, i.e., virtualization of disk space.
iv) The operating system implements virtual devices out of the physical devices because it is cumbersome to work with physical devices and virtual devices prepare a simple interface instead.

In this way, virtual machine or extended machine (in the form of virtual processors, virtual memory, and virtual devices) is created from the physical computer. On a single physical machine, multiple virtual computers are created in the form of multiple processes (see Fig. 1.13). Each user has an illusion that he is using a single machine. Therefore, operating system is also viewed as a manager of the virtual machine or extended machine.

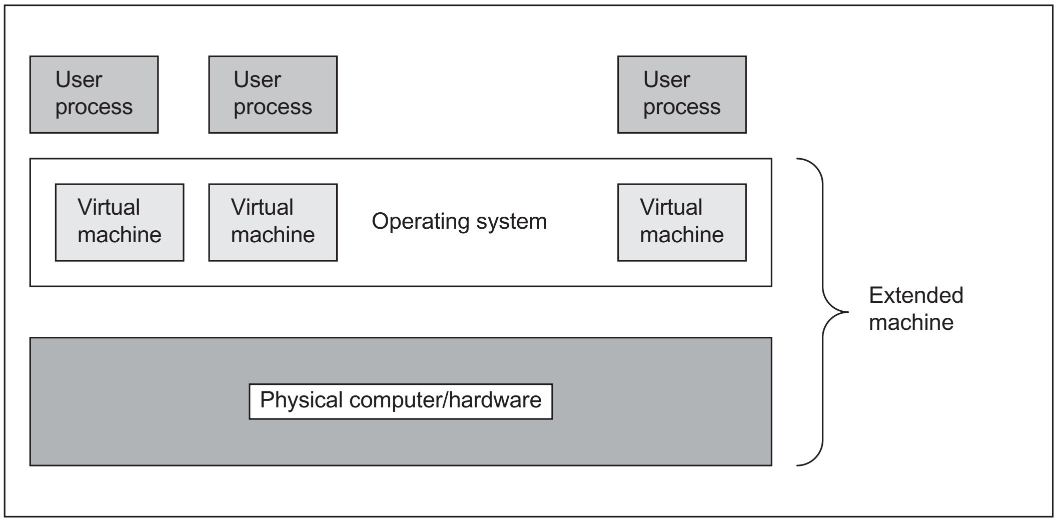


Fig. 1.13 Operating system as a virtual machine manager

1.7 OPERATING SYSTEM'S GENERIC COMPONENTS

In this section, a generic structure of operating system is discussed. The detailed structure and various types will be discussed later. The emphasis here is to know how the interfaces between user, operating system, and hardware are in place. The reader should be aware that the user or programmer cannot access hardware resources directly. Even, in the subsequent chapters a concept will be established that no user is allowed to perform any I/O operations. Therefore, it becomes necessary to know the generic structure of an operating system.

In Fig. 1.2, we have seen the computer system where operating system fits therein. We have seen there that operating system is the interface between user's applications and hardware. It means that whatever job a user wants to perform through the hardware of the computer system will be performed by operating system on behalf of the user. But there is a question of how to tell the operating system the functions we want to perform. It means there should be an interface by means of which user tells the operating system to perform operations on the hardware. This interface is the place where the users give the commands through control statements. There is a program which reads and interprets these control statements and passes the signals to operating system. This program is known as *command-interpreter* or *shell*. Thus, there is a clear separation between the user application, OS, and hardware as shown in Fig. 1.14. The shells may be in the graphical form wherein commands are in the form of mouse-based window and menu system as used in Windows OSs or commands form wherein commands are typed in by the user as used in MS_DOS or UNIX operating systems.

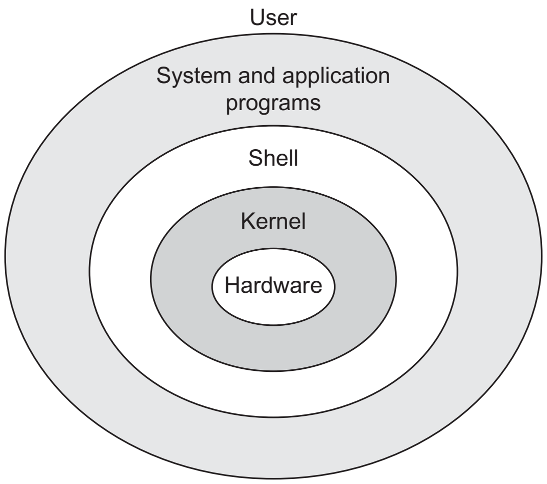


Fig. 1.14 Operating system structure with shell and kernel

As the requirements have grown, the size of the operating systems has also increased. But we know that it needs to be loaded into the main memory which is already packed with user programs. Therefore, the operating system to be loaded into the memory should be of smaller size otherwise most of the memory will be taken by the operating system only. Therefore, essential modules of the operating system such as task management, memory management, etc. are only loaded into the memory known as *kernel*. The kernel is the innermost layer close to the hardware to get things done. Other modules of operating system are stored in the secondary storage like hard disks and get loaded as and when required. For example, virtual memory module is not part of kernel but will be loaded if required. In this way, the operating system part is also divided into two parts: essential part (kernel) and secondary part.

SUMMARY

There was a time when a user on the computer system used to get the program executed in days because everything for program execution was manual and in fact the user was close to the machine. But with the advancement in the technology, a software layer between the user programs and hardware was added so that the user is relived from the details of the hardware and all the work related to machine was done via this software layer. This software layer is known as operating system. The OSs evolved with the increase in demands of the user and inventions in computer hardware and I/O devices. The advancements in computer architecture have always impacted the development of OSs. But sometimes, the researchers of OSs also demanded to have modifications in the architecture. Thus, OSs and architecture both have affected each other and developed in parallel.

Multi-programming is a central concept in operating systems. The multi-programming, i.e., placing more than one program in the main memory, has given birth to other modules of operating system. In fact, the multi-programming originated many problems. As a solution to these problems, other modules of operating system were developed. For example, multi-programming demanded that memory should be partitioned and allocated to the required processes. All the processes must be protected. Multiple processes will compete for limited I/O devices. Multiple processes must communicate and synchronize with each other. Therefore, memory management, process management, process scheduling, device management, process communication, process synchronization, protection, etc., have been developed in response to the problems of multi-programming. All these concepts are relevant to a designer. For a system designer, the operating system is a resource allocator, extended machine manager, and control program. As a resource manager it allocates and manages the resources in the system. As an extended machine manager, it acts as an extended machine in support of the actual hardware and seems to a general user that all the facilities have been provided by the machine hardware only. As a control program, the operating system protects all the programs and itself from any malicious job.

However, all these concepts are not related to the user. A general user's view is different from the system's view. The user wants the convenience while working on the system. There are many facets of the user convenience. The user does not want to indulge into the hardware details. The user wants the interaction with his job so that he can debug it. The user does not want to work with the commands. He wants the GUI based flexibility and convenience. And all these have been incorporated in the operating systems. Thus, the prime goal of an operating system is to have the user convenience so that there is a friendly environment on the system for the user. The other goal of the operating system is the utilization of the hardware and all other resources in the system.

Let us have a quick review of important concepts in this chapter:

- An OS is a software that acts as an interface between the users and hardware of the computer system.
- An OS is a software that provides a working environment for the users' applications.
- An OS is a resource manager that in background manages the resources needed for all the applications.
- Multi-programming is the central concept in operating system that originates all other concepts of operating system.
- Multi-programming places more than one job/program/task in the main memory.
- Multi-programmed batch systems place more than one jobs/programs/tasks in the main memory of a batch prepared for same type of jobs and execute them by switching between them.
- Multi-user systems place more than one job/program/task in the main memory of the main computer system. The jobs are of different users who are connected through terminals to the main computer. The jobs are scheduled by time-sharing technique.

- Multi-tasking systems place more than one job/program/task in the main memory of the system. The jobs here are of a single user working on the system. The jobs are scheduled by time-sharing technique.
- The primary goals of operating system are convenience of the user and best utilization of the hardware.
- There are two views to look at the functioning of the operating systems: user view and system view.
- From the user's viewpoint, the operating system acts as an easy interface between the user and computer system and presents a friendly environment wherein the user can work efficiently without worrying about any configuration or details of the hardware.
- From the system's viewpoint, the operating system acts as a resource manager, control program, and virtual machine manager.
- As a resource manager, operating system schedules and manages the allocation of all resources in the computer system.
- As a control program, operating system controls the user activities, I/O access, and all other activities performed by the system.
- As a virtual machine manager, operating system provides a layer on the actual hardware on which it performs the tasks of the user. And to the user, it seems that all the work done is by the hardware. In other words, there is an illusion created by the operating system that there is a virtual machine which is performing all the work.
- There are two generic components of operating system: shell and kernel.
- Shell is a program which reads and interprets the control statements entered by the user to perform a task. It is also known as command interpreter.
- Kernel is the part wherein only essential modules of the operating system are placed.

MULTIPLE CHOICE QUESTIONS

1. Automatic job sequencing is performed by ________.
   (a) operating system (c) job pool
   (b) resident monitor (d) none
2. Disks were invented in ________ generation.
   (a) first (c) third
   (b) second (d) none
3. SPOOL is ________.
   (a) simultaneous printer operation offline
   (b) simple peripheral operation offline
   (c) simultaneous peripheral operation offline
   (d) simultaneous peripheral operation online
4. MULTICS is
   (a) multiplexed information control system
   (b) multiple input control system
   (c) multiplexed information and computing service
   (d) none
5. PDP-7 was a ___.
   (a) mini computer (c) PC
   (b) Mainframe (d) none
6. IBM with Bill Gates hired Tim Paterson who had written one OS known as
   (a) UNIX (c) Windows
   (b) DOS (d) none
7. Batch systems were developed in ____ generation.
   (a) first (c) third
   (b) second (d) none
8. Spooling was developed in ____ generation.
   (a) first (c) third
   (b) second (d) none
9. Time-sharing was developed in ____ generation.
   (a) first (c) third
   (b) second (d) fourth
10. Multi-tasking/Multi-threading was developed in ____ generation.
    (a) first (c) third
    (b) second (d) fourth
11. _______processing is largely useful in defence applications.
    (a) Batch (c) Parallel
    (b) Real-time (d) None
12. Symbian OS is used in ____.
    (a) smartphones (c) Palm pilot
    (b) smart cards (d) none
13. When a user wants to open and work simultaneously on many windows on his system, what OS should be chosen?
    (a) Multi-user OS (c) Batch OS
    (b) Multi-tasking OS (d) Networked OS
14. When a user wants to remotely log on to a system, wants to transfer a file, etc., on a network system, what OS should be chosen?
    (a) Multi-user OS (c) Batch OS
    (b) Multi-tasking OS (d) Networked OS

15. When computational speed and resource sharing is required and implemented through various full computer systems in a network, what OS should be chosen?
    (a) Real-time OS (c) Embedded OS
    (b) Distributed OS (d) Networked OS
16. What OS should be chosen which is applicable to systems that require time-bound response?
    (a) Real-time OS (c) Embedded OS
    (b) Distributed OS (d) Networked OS
17. What OS should be chosen which will be used in consumer electronics items, mobile phones, smart cards, etc.?
    (a) Real-time OS (c) Embedded OS
    (b) Distributed OS (d) Networked OS
18. Program which reads and interprets these control statements and passes the signals to operating system is known as
    (a) system programs (c) shell
    (b) system call (d) kernel
19. _______is the innermost layer close to the hardware to get things done.
    (a) System programs (c) Shell
    (b) System call (d) Kernel
20. *Apple Macintosh* was commercially successful not only due to its cheaper cost but also because it was_____.
    (a) taking less memory (c) accessing I/O faster
    (b) user friendly (d) none

REVIEW QUESTIONS

1. What is the need for an operating system?
2. What are the functions of an OS from user's viewpoint?
3. What are the functions of an OS from system's viewpoint?
4. What were the difficulties in second generation from OS viewpoint?
5. What is a resident monitor?
6. What is JCL?
7. What is offline operation?
8. What is the difference between online and offline operation on a computer system?
9. How did the disks solve the problem faced with the magnetic tapes?
10. What is SPOOL? What is the benefit of spooling?
11. Give a brief overview of development of UNIX?
12. Explain the difference between DOS, UNIX, Apple Macintosh, and Windows?
13. Explain the differences between multi-programming, multi-user, and multi-tasking OSs.
14. Explain the characteristics of multi-processor and distributed systems.
15. What is the differences between network and distributed OSs?
16. What is the difference between real-time and embedded operating systems?
17. How does operating system function as resource manager?
18. How does operating system provide protection?
19. What is a virtual machine? How does operating system function as a virtual machine manager?
20. Discuss the role of shell and kernel in operating system.
21. What are the challenges in designing a multiprocessing/distributed operating systems?
22. What is the difference between a smart card and smartphone?

BRAIN TEASERS

1. Can you work without operating system in your computer system?
2. The major drawback of multiprogrammed batch systems was the lack of user/programmer interaction with their jobs. How can you overcome this?
3. The response time is the major requirement of a multiuser time-sharing OS. What are the things that need to be improved for this requirement from a system designer's viewpoint?
4. Is time-sharing OS suitable for real-time systems?
5. Examine the following conditions and find appropriate operating system for them:
    (a) In a LAN, users want to share some costly resources like laser printers.
    (b) Multiple users on a system want quick response on their terminals.
    (c) Railway reservation system
    (d) A user wants to work with multiple jobs on his system.
    (e) In a network system you want to transfer file and log on to some node.
    (f) There are some jobs in the system which does not want user interaction.
    (g) Washing machine
6. Explore the features of operating system being used in recent design of smartphones.

7. Do all operating systems contain shell?
8. Multi-programming is inherent in multiuser and multi-tasking systems. Explain how.
9. There are four jobs running in a multi-programming system with the following requirements:

   job 1: requires disk after every 1 min, device service time including wait and access = 3 min, total processing time = 4 min.

   job 2: does not require any I/O, total processing time = 7 min.

   job 3: requires printer after every 3 min, device service time including wait and access = 2 min, total processing time = 9 min.

   Prepare a timing chart showing the CPU and I/O activities of the jobs. Compute the total time for execution using monoprogramming and multiprogramming and then compare the results.

2 Hardware Support for Operating Systems

2.1 INTRODUCTION

The first chapter introduced the basic concepts of an OS. Before we delve into the details of an OS, the knowledge of computer system architecture is a prerequisite to understand the concepts of an OS. Since there was a parallel development in computer architecture and the OSs as we have seen in Chapter 1, it is necessary to understand the relation of architecture with OS. Therefore, some basic concepts that are related to OS have been discussed in this chapter. Since the modern OSs are interrupt driven, the interrupt mechanism has been explained. The protection among the user jobs and the OS is a major issue to implement the multi-programming-based concepts in OSs. Therefore, it is necessary to understand how the protection has been achieved in the hardware. The management of I/O devices is a major area where the OS plays a great role. All the fundamental issues related to I/O devices such as type of devices, device controllers, and the device drivers have also been discussed. The magnetic disk is a widely used secondary storage device and used in many concepts of OS such as scheduling, virtual memory, and so on. Therefore, the structure of the disk and its related issues have also been discussed.

2.2 INTERRUPT-DRIVEN OPERATION FOR OPERATING SYSTEM

The modern OSs that support the multi-programming/multi-user/multi-tasking environment perform interrupt-driven operation, i.e., everything an OS does is interrupt driven. If there is no event and no processes to execute, the OS does nothing. It simply waits for an event. The OS is activated when there are processes to execute or an event causing the interrupt. Therefore, it is important to understand what an interrupt is and what happens to the processor when an interrupt arrives. So, let us discuss the concept of interrupt.

Interrupt is a signal sent by hardware or software to notify the processor about the occurrence of an event that needs immediate attention. On the processor hardware, there is an interrupt request (IRQ) line that the processor senses for any interrupt after execution of each instruction of the process. If there is no interrupt, it moves to next instruction to execute. But if there is an interrupt, the state of the process being executed is saved so that the processor can resume its execution from the place where it left off (see Fig. 2.1).

Learning Objectives

After reading this chapter, you should be able to understand:

- Interrupts, their types, and interrupt-driven operation of an OS
- Types of I/O devices
- Introduction to timers
- Role of device controllers and device drivers
- Multiple mode of protection
- Input-output protection
- Memory protection
- CPU protection
- Input-output communication techniques
- Structure of a magnetic disk
- Disk partitioning
- Disk formatting

```
while (fetch next instruction)
{
    Execute the instruction;
    If (there is an interrupt)
    {
        Save the state;
        Find address of ISR;
        Execute ISR;
        Return from ISR and restore the state;
    }
}
```

Fig. 2.1 Interrupt view of processor

After saving the state of the old process, a program known as an *interrupt handler* or *interrupt service routine* (ISR) is executed, which is a part of microcontroller firmware (such as ROM-BIOS that provides a small library of basic input/output functions used to operate and control the peripherals such as the keyboard, display, disk, etc.), OS or a device driver. There is an ISR corresponding to each interrupt generated. After executing the ISR, the control is returned to the interrupted program and its execution is resumed by loading its saved state.

But we do not know the address of an ISR to be executed. The addresses of all ISRs are placed in a list known as *interrupt vector table* (IVT). The IVT is generally placed in low memory. Each interrupt has a unique number and therefore in IVT, corresponding to an interrupt number, the address of the ISR is stored. Whenever an interrupt is sensed by the processor, it finds out its number and the address of the ISR in IVT. After finding the address of the ISR, the control is transferred to the ISR and it is executed. In the x86 architecture, each address in the IVT is 4 bytes long and supports 256 total interrupts (0-255). To access the location of an interrupt in IVT, the interrupt number is multiplied by 4 as each address is 4 bytes long. For example, hitting a keyboard generates a hardware interrupt whose number is 9. It means the address of ISR corresponding to this interrupt will be found on locations 36, 37, 38, and 39.

The steps of the interrupt processing are summarized as follows (see Fig. 2.2):

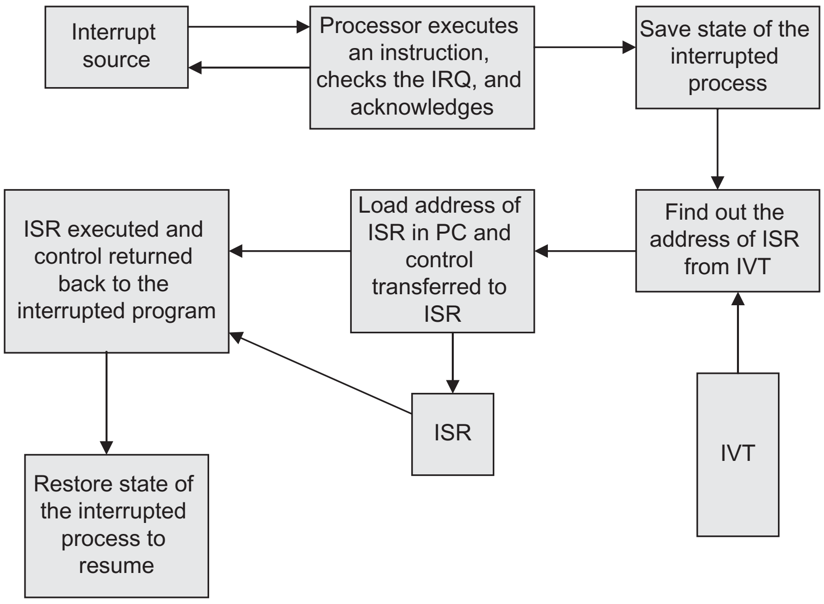

Interrupt source
Processor executes an instruction, checks the IRQ, and acknowledges
Save state of the interrupted process
Find out the address of ISR from IVT
Load address of ISR in PC and control transferred to ISR
ISR executed and control returned back to the interrupted program
ISR
IVT
Restore state of the interrupted process to resume

Fig. 2.2 Steps of interrupt processing

1. The interrupt is generated from its source (hardware or software).
2. The interrupt signal generated is pending on the interrupt request (IRQ) line of the processor hardware.
3. The processor finishes its current instruction execution and then checks for the interrupt.
4. The processor determines that there is a pending interrupt request and sends an acknowledgement signal to the source that generated the interrupt.
5. After the acknowledgement, the source of the interrupt removes the interrupt request signal.
6. The processor saves the current state of the process that was interrupted such as program status word (PSW), program counter (PC), processor registers, and other data structures that will be discussed later in the book. These can be pushed onto a control stack.
7. The processor finds out the type and number of the interrupt generated and finds the address of the corresponding ISR in IVT.
8. The processor loads the address of the ISR in PC and executes it.
9. After executing the ISR, the control is returned back to the interrupted program and saved state is loaded again so that the program can be resumed.

2.2.1 Types of Interrupts

There may be various sources of interrupts. In general, there may be two types of interrupts as follows:

Hardware Interrupts

A hardware interrupt is generated from an external device, which could be either a part of the computer itself such as a keyboard, disk, or an external peripheral. For example, when we press a key on the keyboard or move the mouse, the hardware interrupts are triggered, which in turn causes the processor to read the keystroke or mouse position.

Software Interrupts

A software interrupt may be generated due to the following:

- There may be conditions caused within the processor that require OS attention. For example, if there is an arithmetic exception like divide-by–zero during the execution of a process.
- There are some instructions in the user process which are treated as a request to the OS. These instructions, known as privileged instructions, are the medium through which a user process can indirectly interact with hardware through the OS. For example, if the user process wishes to read input data from the keyboard, then the user process will use a privileged instruction that will be passed to the OS and treated as a software interrupt.

Thus, the software interrupts are the result of an exceptional condition in the process or may be caused due to a special instruction in the instruction set that triggers the interrupt when executed. The exceptional condition is sometimes known as a *trap*. In general, it is used for errors or events occurring during the program. The maximum number of hardware interrupts that can be handled depends on the number of IRQ lines to the processor. However, the software interrupts are not limited to the number of IRQ lines and therefore can be hundreds in number.

2.2.2 Multiple Interrupts Handling

It is not so that only one interrupt may arrive at a time. We will see later in the exploration of multi-programming concept that multiple interrupts may also arrive and require the processor's attention. For example, in a multi-programming environment, a printer is printing the output, the keyboard is receiving the input, and the data is read from the disk. All these events cause interrupts. However, the processor is able to execute one interrupt at a time. There are two approaches to solve this problem. One is to disable the other interrupts while the ISR corresponding to one interrupt is being executed. The interrupts arrived during the execution of the ISR are treated as pending and may be stored in FIFO queue. Once the ISR execution is over, the other interrupts in the queue are processed. But the disadvantage of this approach is that some interrupts which need immediate attention and not get serviced may do some loss. For example, if an ISR is being executed and the keyboard interrupt arrives that is reading some data. If keyboard interrupt is not processed immediately, the input data may be lost. Therefore, the second approach—a priority mechanism—is taken that decides the priority of the interrupts arriving. On the basis of the priority decided, the interrupts are serviced. If a lower priority ISR is being executed and a higher priority interrupt arrives, the ISR is interrupted and the control is passed to the high priority ISR. After the execution of this high priority ISR, the control is returned to the older ISR. Even if no ISR is being executed and two interrupts arrive at the same time, the interrupt with higher priority is executed first. In this way, the priority based interrupt mechanism is used to handle multiple interrupts.

2.3 INPUT–OUTPUT DEVICES

Since the OS provides a generic, consistent, convenient, and reliable way to access I/O devices, a brief introduction of I/O devices is provided in this section. There are various types of devices available today. They may vary depending on their operation, speed, data transfer, etc. Some devices may be categorized based on these characteristics. Even within a category the devices may vary. This is the reason that device management is necessary as part of operating system function. The devices may be categorized as human readable and machine readable. The human readable devices are mouse, keyboard, monitor, etc. The machine readable devices are sensors, controllers, disks, tapes, etc.

The devices may transfer the data as a stream of bytes or in the form of a block. If the device accepts and delivers the data as a stream of characters/bytes, it is known as character device. *Character-oriented devices* are suitable where linear stream of bytes are required. For example, while accepting input data from the keyboard, a block of characters cannot be expected in one instance. Therefore, input devices like keyboard, mouse, modems, etc., are all examples of character devices. Even the output devices of this nature, like printers, are also character devices. On the other hand, if the device accepts and delivers the data as a fixed sized block, it is known as *block-oriented device*. Disk is the example of a block device. Another criterion is how a device accesses the data. On the basis of accessing data sequentially or randomly, the devices are called as *sequential device* such as a tape drive and *random access device* such as a disk.

A different type of I/O device is the *network device*. The network devices differ from conventional devices like disk in the sense that they need special I/O interfaces to send or receive data on a network. For example, socket is the major I/O interface used for network devices.

There may be two types of I/O devices: blocking and non-blocking. In blocking devices, the program is blocked with an I/O action and is not allowed to execute the program until the I/O action completed. For example, the word processor program waits for a key press or a mouse click (I/O) done by the user and then starts processing. In non-blocking devices, the device is checked periodically for an I/O. If there is a process that processes the data and displays it on the screen but needs to check the I/O on keyboard and mouse as well. In this case, the process periodically checks the keyboard and mouse for I/O while processing and displaying the data. In another example, the video application reads data from a file and simultaneously decompressing and displaying the data on the screen.

Other criteria to define the types of devices may be based on complexity of control, data representation, error conditions, etc.

2.3.1 Timers or Clocks

The timer is an important device that is used in the operating system to implement multi-tasking and other jobs. It is used to have the current time and elapsed time and to trigger some operation at a particular time instant. The timers are used for the following purposes:

- The periodic interrupts may be generated.
- It may be used by a scheduler to generate an interrupt when there is need to preempt a process when its time slice expires.
- It may be used by a disk subsystem when there is need to flush the modified cache buffers to the disk.
- It may be used to cancel the operations in a network that are causing congestion and therefore taking a long time to process.
- It may be used to cancel the operation of a process that is not releasing the processor and holds it for a long time. It helps in sharing the processor time among multiple tasks and every task gets fair time and no task holds the processor.

A timer is implemented with a hardware known as up-counter that counts incoming pulses. A counter acts as a timer when the incoming pulses are at a fixed known frequency. For example, the programmable interval timer (PIT) hardware is used for the function of a timer.

A timer consists of the following components:

- Pre-scaler
- N-bit timer/counter register
- N-bit capture register

The pre-scaler component allows the timer to be clocked at the rate we wish. It takes the basic timer clock frequency (may be the CPU clock frequency or some higher or lower value may also be taken), divides it by some value, and then feeds it to the timer. The timer register (an up-counter) reads and writes the current count value and may stop or reset the counter. The regular pulses which drive the timer are called *ticks*. Thus, a tick is a basic unit to express the timer value. When there is some event, the current count value is loaded in the capture register. Besides these components, a compare register is also used that holds a value against which the current timer value is continuously compared. When the value in timer register and the value in compare register matches, an appropriate event is triggered.

2.4 DEVICE CONTROLLERS

A device controller, also known as an *adapter*, is an electronic device in the form of chip or circuit that controls the communication between the system and the I/O device. It is inserted into an expansion slot of the mother board. It functions as a bridge between the device and the operating system, i.e., the operating system deals with the device controller of the device to which it wishes to communicate. It takes care of low level operations such as error checking, data transfer, and location of data on the device. Each device controller is designed specifically to handle a particular type of device but a single controller may handle multiple devices also.

To perform an I/O operation on a device, the processor sends signals to its device controller. For example, it may be the case that the data needs to be read from a serial device. So the processor sends a command to the device controller of that serial device first to read the desired bytes of data. In turn, the controller collects the serial bit stream from the device and converts it into a block of bytes. It may also perform some necessary error corrections if required. There is a buffer inside the controller that stores the block of bytes thus obtained. After this, the block of bytes from the buffer of controller is copied to the memory. Now if these data need to be displayed, the device controller for the display reads the data from the memory and sends the signal to the CRT to display the data. In this case, the operating system initializes the device controller with required information such as address of the bytes to be read, number of characters per line, and the number of lines on the screen to be displayed.

For the purpose of communication with the processor, each device controller has a set of following device registers (see Fig. 2.3):

Control Register

These are used by the processor to configure and control the device. This register is meant to write the data, i.e., the processor can alter but not read them back.

Status Register

These registers provide the status information about an I/O device to the processor. This register is meant to be read-only, i.e., the processor can only read the data and is not allowed to alter.

Data Register

This register is used to read and write data from/to the I/O device.

The operating system performs I/O operations with the use of these registers only by sending commands to an appropriate register. The parameters of the commands are loaded first into the

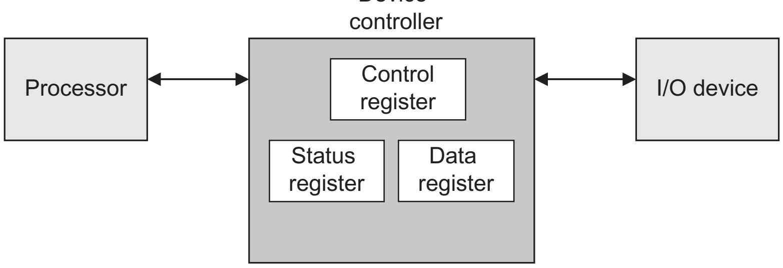


Fig. 2.3 Device controller registers

controller's registers. When the command is accepted, the control is passed to the controller by the processor. When the control is passed to the controller, the processor is free to do other job during this time. When a command has been completed, the controller triggers an interrupt to inform the operating system that the desired operation has been completed. The operating system after gaining the control again gets the result of the operation and checks device status by reading information from the controller's registers.

2.5 DEVICE DRIVER

The most challenging task for an operating system is to manage the I/O devices in a computer system. It acts as an interface between devices and computer system. This interface should be simple, easy to use for a user, and preferably same for any type of device. However, today there are a myriad of input and output devices. Each I/O device has its own detail and complexity. In this case, operating system needs to be changed to incorporate every newly introduced device. Therefore, the I/O functionalities should be treated separately in the operating system so that the other parts of the operating system are not affected. The software which deals with the I/O is known as *I/O software*. In I/O software, there are two types of modules. First module deals with the general functionalities when interfacing with any type of device, i.e., these functions are common while interfacing with any I/O device and are known as device-independent I/O software. For example, there should be a general interface for any type of device. The second module provides device-specific code for controlling it and is known as device driver. The second module in fact takes care of the peculiarity and details of a particular device, i.e., how to read or write data to the device. In this way, operating system does not need to change its code again and again to incorporate any new device. Its I/O software takes care of all the I/O devices to be interfaced with the system without changing the OS code.

Each device needs a device-specific code in the form of device driver for controlling it. As discussed earlier, each device has a device controller that has some device registers for performing I/O operations on the device. But the number of device registers and the nature of commands for performing I/O operations vary from device to device. Therefore, to communicate with each type of device controller, a specific code in the form of a device driver is written, which takes care of the specific device controller registers and the commands. Thus, the device drivers act as a layer that hides the differences among the device controllers.

The device drivers are part of the operating system, but not necessarily part of the OS kernel. These are software modules that help the operating system such that there is easy access to the hardware. They need to be installed on the system for each device we need to use. In general, the manufacturer of the device supplies the device drivers. However, the device drivers may differ according to the operating system type and its version.

The device driver communicates with the device controllers and thereby with the device with the help of interrupt-handling mechanism. When the device controller interacts with the actual device, the data are transferred between the actual device and the controller according to the I/O operation, i.e., the data from the device are written to the controller's register in case of input operation or the data from the controller's register are sent to the device in case of output operation. After completion of I/O operation at the level of device and device controller, the device controller generates an interrupt to the device driver

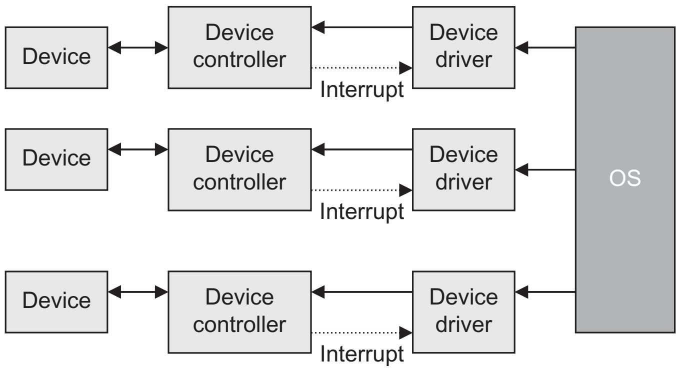


Fig. 2.4 Functioning of device driver

(see Fig. 2.4). The interrupt service routine is executed in order to handle a specific interrupt for an I/O operation. This routine extracts the required information from the device controller's register and performs the necessary actions. After the completion of an ISR, the blocked device driver is unblocked and may run again.

2.6 PROBLEMS IN THE DESIGN OF MODERN OSs

When the multi-programming concept was introduced, a new generation of OSs was evolved. The modern OSs have multi-programming as an inherent concept. But when the multi-user time-sharing and multi-tasking concepts were developed, many problems arose for their implementation. To implement them, there was no architectural support. For example, the Intel microprocessor series till 80186 was not able to support the multi-user and multi-tasking concepts. Let us first discuss the problems occurred:

- Since the multi-programming concept allows the switching between the processes, there was a need to preserve the state of the process that was stopped temporarily so that it can be resumed when the processor switches back to it. Similarly, the state of the process where the processor switches currently needs to be loaded. In this way, a mechanism is needed to save and load the state of a process.
- Since the resources are limited as compared to the number of processes in a system, the processes need to share them. There are several problems due to this environment. One of them is that the processes may try to access the devices at the same time. There should be a mechanism so that the processes have an orderly access to the devices.
- The multiple processes sometimes may be trapped in a deadlock situation while accessing the resources. Suppose there are two processes P1 and P2 and two resources R1 and R2 in a system. P1 is using R1 and needs to have R2 to continue its execution. But R2 is used by P2 which needs to have R1 to continue its execution. In this situation, both the processes are waiting for each other to release the resource to continue thereby causing a deadlock situation.
- Another problem in multi-programming environment is that all the processes may try to update the contents of a memory location at the same time.
- Since all the processes and operating system reside in the main memory, a process may try to access the memory locations of another process or even access the operating system area in the memory. It may corrupt the operating system or some process.

- A process may engage the processor for an infinite time by having such instructions in it and does not release it. In this case, the other processes will be in wait for that process to release the processor.
- A process while accessing any I/O device may do any illegal operation on it thereby damaging the devices. For instance, in disk operating system (DOS), there is no protection of devices from the user programs. A user may write some virus programs and do some mischievous operation on the devices, e.g., infecting the boot disk by loading the virus program in boot sector, jamming the printer, etc.

All the problems discussed earlier were faced in the design of the multi-programming-based operating systems. The single-user operating systems like DOS were not able to provide the solutions to these problems.

2.7 NEED FOR PROTECTION

Some of the problems discussed earlier may be solved with the help of software support from the operating system. For example, the solution to deadlock or accessing the same memory location is provided by the operating system through deadlock avoidance or detection algorithm and semaphore, respectively. But there are some issues which may not be implemented without the architectural support. Since the problems discussed earlier largely address one problem, i.e., the protection, the processor architecture of that time (e.g., Intel 8086, 8088, 80186) was not able to provide any kind of protection. Any user was able to write a program that might access the memory area of operating system and corrupt it. Any user was able to enter in the memory area of any other user area. These problems initiated the demand for a mechanism that the processes and even the operating system were not protected as there was no provision to prohibit the user from illegal accessing of memory area or I/O device. Further, memory areas divided among various processes and the operating system were not protected.

After this demand for protection, various architectures were developed to incorporate the protection mechanisms such that the multi-programming-based operating system could be implemented. Thus, this protection demand emerged from the operating system implementation need and the result was the new processor architecture. The Motorola MC68000 family of microprocessors, AT&T UNIX operating system, and Intel 80286 and its other derivatives are examples of the modified processors that considered the protection need in multi-programming operating systems.

2.8 HARDWARE PROTECTION FOR MULTI-PROGRAMMING/ MULTI-TASKING

To address all the problems described, various architectural support/modifications taken are discussed as follows.

2.8.1 Multiple Modes of Operation

The basic idea in implementing the protection feature is to separate the regions of operating system and users in order to protect the operating system and hardware devices from damage by any malicious user/program. The modern operating systems separate code and data of the operating system from the code and data of the user processes. This separation was termed as dual mode operation. This dual mode operation has two modes: the kernel mode and the user mode. The processor now can execute in one of the mode at a time, either in the kernel mode or in the user mode. The contemporary processors implement this by having a mode bit in the *program status word*

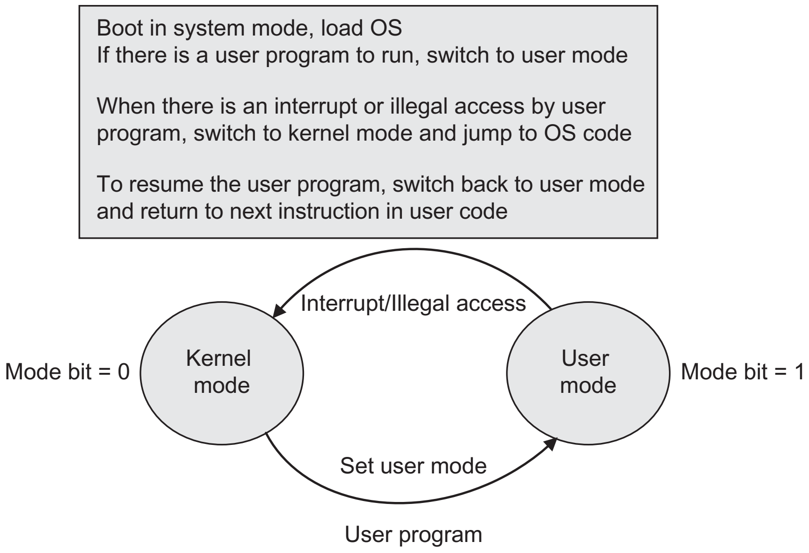

Boot in system mode, load OS
If there is a user program to run, switch to user mode
When there is an interrupt or illegal access by user program, switch to kernel mode and jump to OS code
To resume the user program, switch back to user mode and return to next instruction in user code
Interrupt/Illegal access
Mode bit = 0
Kernel mode
User mode
Mode bit = 1
Set user mode
User program

Fig. 2.5 Mode switching

(PSW) to specify whether the processor is executing in kernel-mode code or user-mode code. The PSW is a collection of control registers in the processor. The control registers control the operation of the processor itself. Initially, the mode bit is set to 0, thereby meaning that the control is with the operating system when the computer system is started. When a user process wants to gain the control, the mode bit is set to 1 and the user is able to execute in his own area but prevented all access to the kernel memory space. However, if a user attempts to access any illegal memory area or instruction, the processor generates an illegal access exception and the mode is switched to the kernel mode. Similarly, if a user wants to access any hardware, the mode is switched from the user to kernel mode. The mode switching is shown in Fig. 2.5.

Since the operating system has more privilege over user processes, the kernel mode has high privilege as compared to user mode. This is why the kernel mode is also called *privileged mode*. The kernel mode is also known as *system mode*, *monitor mode*, or *supervisor mode*. Thus, the system is booted first with the kernel mode and the operating system has all the access to the hardware, thereby initializing a protected environment.

There are multiple levels of protection known as *privilege rings* or *levels*. For example, the MC68000 processor was designed to have two privilege rings as dual mode. The AT&T UNIX was designed with three levels: kernel, shell, and the application. The kernel here is the innermost level and the application is on the outermost level.

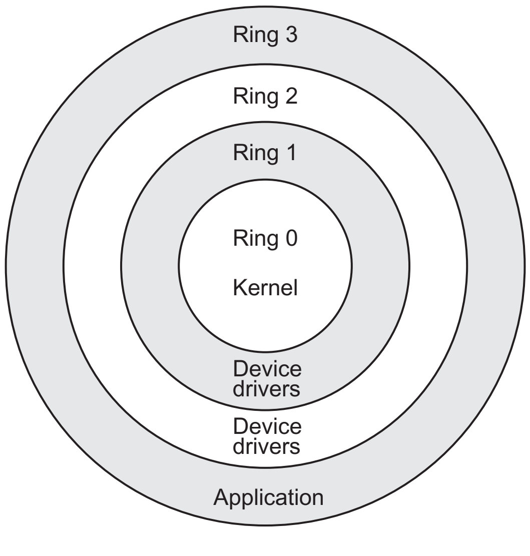


Fig. 2.6 Intel privilege rings

The Intel modern processors come with four privilege rings (0–3) as shown in Fig. 2.6. In this architecture, the operating system is in the innermost level (most trusted) having the highest level of privilege and protected. The outermost level (least trusted) is application level having the least privilege. The other two levels are for device drivers having the high privilege as compared to the application but less privileged than the OS.

2.8.2 Input–Output Protection

The multiple mode operation of the system enhances the security between the user processes and operating system. But, the users should also be prohibited to access the I/O devices directly. Therefore, all I/O instructions are privileged and the privilege to access the devices is with operating system only. It means no user process can access the I/O device directly. To access any I/O device, the process may request the operating system through a *system call*. The system call is a user request to the operating system which is interpreted and executed on the hardware by the operating system on the behalf of the user. In this way, all I/O instructions are privileged thereby providing another level of security. In this sense, the instructions are divided into two parts: *privileged instructions* and *unprivileged instructions*. The illegal instructions mentioned in previous section are unprivileged instructions only. The user process cannot execute privileged instructions. In fact, whenever there is a system call in the user process, the control switches from the user mode to the kernel mode thereby transferring the control to the operating system. After servicing the system call, the mode is again switched back to the user mode and control is with the user process again (see Fig. 2.7). This privilege mechanism with mode switching ensures that the user never gains control to access the devices directly, thereby protecting the I/O devices.

The system call being used in a user process is basically a software interrupt. As a result of this software interrupt, the control is passed to an appropriate interrupt handler and the mode is switched to the kernel mode. The operating system determines and verifies the details of the interrupt occurred, executes the interrupt handler, and then returns the control back to the user by switching mode to user mode.

2.8.3 Memory Protection

Besides the earlier-mentioned protection, a user program can still access the memory region of some other user process. It means the user processes are not protected from any illegal access by some process or any malicious process. Moreover, a user process may access the IVT in memory and may change the address of any interrupt handler and do some illegal operations on the devices. There should be some mechanism to protect the memory regions of each process as well as the operating system. For this kind of protection, each process must know its boundary of execution, i.e., there should be a start address and a limit address that defines the boundary of each process. This was supported by the architecture in the form of *base register* and *limit register*. Each process has defined limits to its memory space. The start address of a process is stored in the base register and the maximum size of the process is stored in the limit register. Whenever a process starts executing and references some memory location (say, *m*), it is checked against the base register. If the memory location being referenced is greater than or equal to the base register and less than the addition of base address and limit, then only it proceeds for execution; otherwise it is considered as illegal memory access. In case of illegal access, the control is transferred to the operating system by switching the mode back to the kernel mode as shown in Fig. 2.8. This check for memory protection is done by the hardware for each process. The base

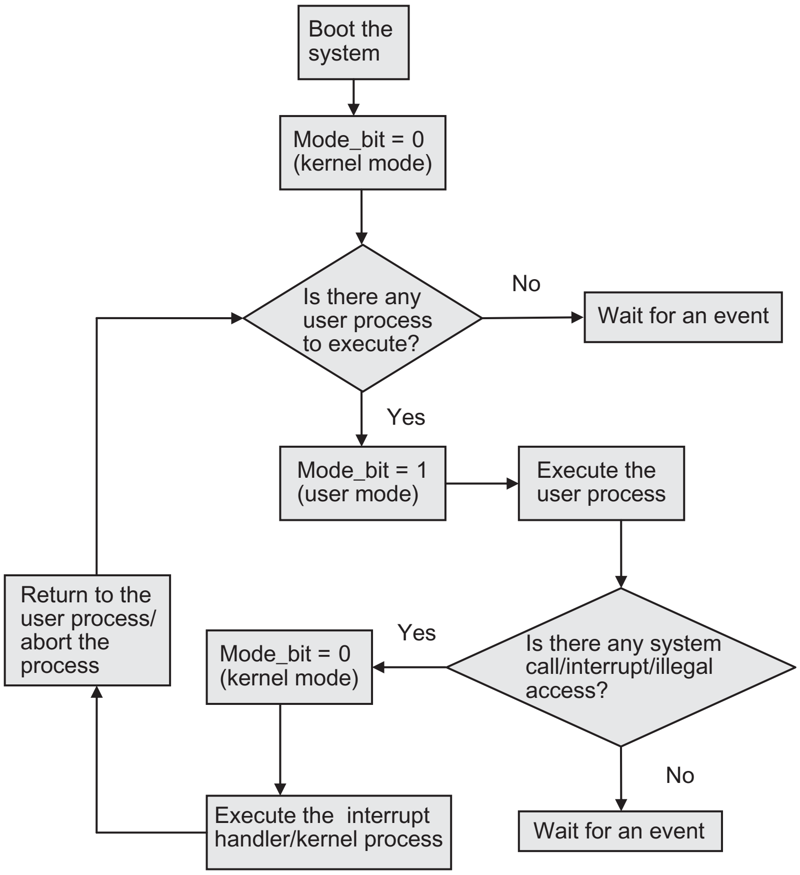

Boot the system
Mode_bit = 0 (kernel mode)
Is there any user process to execute?
No
Wait for an event
Yes
Mode_bit = 1 (user mode)
Execute the user process
Is there any system call/interrupt/illegal access?
Yes
Mode_bit = 0 (kernel mode)
No
Wait for an event
Execute the interrupt handler/kernel process
Return to the user process/ abort the process

Fig. 2.7 I/O protection flow

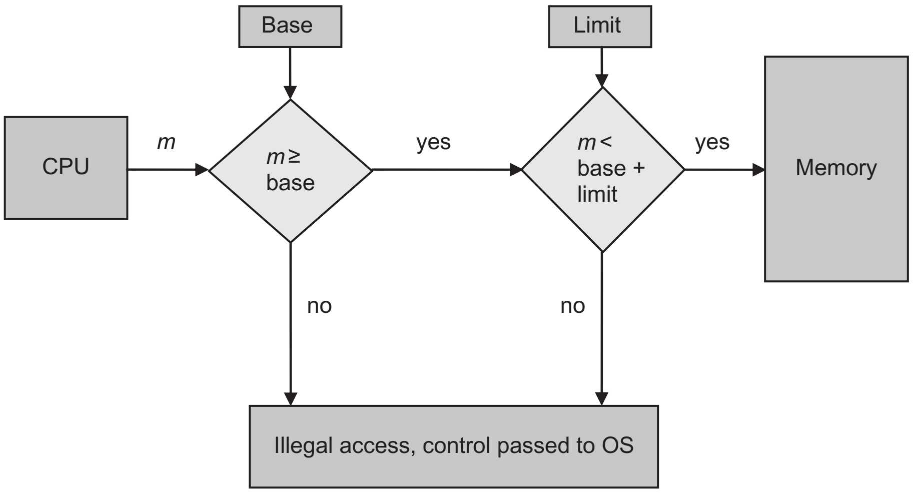


Fig. 2.8 Memory protection

and limit registers are updated for every process (which is to be executed) in the kernel mode by the operating system. This mechanism thus prevents any user process to access or modify another user process or operating system area.

Example 2.1

Figure 2.9 shows the memory structure of some processes and operating system with their legal memory addresses. The base and limit registers are loaded with the addresses 1050 and 1000, respectively. Suppose P1 and P2 reference the memory locations 2040 and 3052, respectively. Check if the processes will be allowed to execute.

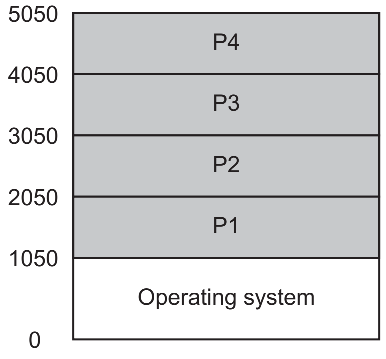


Fig. 2.9 Memory protection

Solution

P1 is first checked against the base register 1050. In this case, the reference memory location of P1, i.e., 2040, is greater than base register. Now, P1 is checked against the sum of base and limit registers, i.e., 1050 + 1000 = 2050. Since it is less than 2050, it will be allowed to execute.

On the other hand, if P2 references a memory location 3052, it is not allowed to execute because it violates the second criterion, i.e., m < limit + base. So the control is passed to the operating system as it attempts to access the memory location of P3.

2.8.4 CPU Protection

There may be some situation that a user process gains the control of the processor and has a set of instructions that are being executed for an infinite time and thereby not relinquishing the control of the processor. Thus, it leads to the situation when the processor is also not safe and must be protected from the user processes. There should be a mechanism such that the processor does not get trapped infinitely in a user process and returns the control back to the operating system. To achieve this, again the hardware support is required. A timer is used that interrupts the processor after a specified period of time. The timer is implemented with the clock and a counter. All the operations related to the timer modification are executed by the operating system as these are treated as privileged operations. The operating system sets the counter for a time period. Every time the clock ticks, the counter is decremented and an interrupt is generated when the counter reaches to 0. On the generation of interrupt, the control is switched to the operating system. In this way, no user process can hold the processor beyond a limit and has to relinquish it after a specified period of time, thereby protecting the processor. The timers are also helpful in implementation of multiuser time-sharing systems. In these systems each user gets a uniform time to execute his process. This is achieved by setting the timer for a fixed period of time and interrupt is sent when the time of a user process expires.

2.9 INPUT–OUTPUT COMMUNICATION TECHNIQUES

There are three techniques by which I/O operation can be performed on a device. These are known as I/O communication techniques. These techniques are used to have a mode of communication between the user request and the device, taking device characteristics into account.

2.9.1 Programmed I/O

Whenever a process is being executed and the processor finds an I/O instruction, it issues the commands to the appropriate device controller. The device controller performs the operation by interfacing to the physical device and then sets the status of the operation in a status register. But this is the job of the processor to check whether the operation has been executed or not. For this purpose, it continually checks the status of the operation until it finds the operation is complete. Therefore, the process is busy waiting until the I/O operation has not been performed.

The I/O operation is performed using a processor register and a status register. The device puts the data in the processor register when input operation is required. On the other hand, the device reads the data from the register when output operation is required. After completion of the I/O operation, the status of the operation is written in the status register as a flag. In this way, the processor executes the instruction in such a way that it is in direct control of the I/O operation, i.e., sensing a device status, sending read/write command to the device, and transferring the data.

There is one disadvantage of this technique that the processor is busy waiting for the status of the operation while the I/O module is performing. At this time, the processor is not executing other instructions of the same process or any other process and is tied up for only one I/O operation. For the I/O operations that consume very less time or the systems where the processor has no other job to do, the programmed I/O is better. But for the multi-tasking environment, programmed I/O is not a better choice where several processes are in queue waiting for the processor.

2.9.2 Interrupt-driven I/O

In programmed I/O technique, the processor time is wasted as it continually interrogates the status of I/O operation. It would be better if the I/O operation is started and the processor switches to another process to be executed instead of waiting. Therefore, the processor issues I/O command to the device controller for performing I/O operation and switches to another processor by calling the scheduler that schedules the process to it. The question is how the processor knows when the I/O is complete. This is done through the interrupt mechanism. When the operation is complete, the device controller generates an interrupt to the processor. In fact, the processor checks for the interrupt after every instruction cycle. After detecting an interrupt, the processor will stop what it was doing by saving the state of the current process and resumes the previous process (where I/O occurred) by executing appropriate interrupt service routine. The processor then performs the data transfer for the I/O operation.

For example, when a user requests a read operation from an input device, the processor issues the read command to the device controller. The device controller after receiving this command starts reading from the input device. The input data from the device needs to be stored on the controller's registers. But it may take some time and this time is sufficient to serve any other process. Therefore, the processor is scheduled to execute any other process in the queue. As soon as the data become available in the controller's register, the controller signals an interrupt to the processor. The appropriate interrupt handler is run so that the processor is able to get the data from the controller's register and save them in the memory.

Since the modern operating systems are interrupt driven, they service the I/O requests using the interrupt mechanism only.

2.9.3 Input/output Using DMA

When a user wants to input some data through the keyboard or some data are printed on the screen after every character to be input or output, the processor intervention is needed to transfer the data between the device controller and the memory. Suppose a user inputs a string of 50 characters length and for every character to input there is 10 millisecond time required. It means between two inputs there is a 10-ms time duration, thereby having an interrupt. It causes to have a number of interrupts just to enter 50 characters long string. Thus, when the data are large, interrupt-driven I/O is not efficient. In this case, instead of reading one character at a time through the processor, the block of characters is read. This operation is known as *direct memory access* (DMA), i.e., without the processor intervention. In DMA, the interrupt will not be generated after every character input. Rather a block of characters is maintained and this block is read or written. So when a user wishes to read or write this block, the processor sends the command to the DMA controller and rest of the responsibility to do I/O operation for one block is given to this DMA controller. The processor passes the following information to the DMA controller:

- The type of request (read or write)
- The address of the I/O device to which I/O operation is to be carried out
- The start address of the memory where the data need to be written or read from alongwith the total number of words to be written or read. This address and the word count are then copied by the DMA controller in its registers.

Let us suppose, we need to perform a read operation from the disk. The CPU first sends the information mentioned above to the DMA controller. The information is stored in the disk controller registers. There are three registers as follows:

- *Memory address register* states the address where the read/write operation is to be performed.
- *Byte count register* stores the number of bytes to be read or written
- *Control register* specifies the I/O port to be used, type of operation, the data transfer unit, etc.

The DMA controller initiates the operation by requesting the disk controller to read data from the specified address and store in its buffer. The buffer data are then transferred to the specified memory location. When this write is complete, the disk controller sends an acknowledgement signal to the DMA controller. The DMA controller increments the memory location for the next byte and decrements the byte count. The disk controller again copies the next byte in the buffer and transfers the byte to the memory. This process goes on until the byte count becomes zero. When the full I/O operation is complete, the DMA controller sends an interrupt to the processor to let it know that the I/O operation has been completed.

Thus, in DMA-based I/O, instead of generating multiple interrupts after every character, a single interrupt is generated for a block, thereby reducing the involvement of the processor. The processor just starts the operation and then finishes the operation by transferring the data between the device controller and memory.

2.10 MAGNETIC DISKS

A magnetic disk is a widely used secondary storage device in the computer system. It consists of a set of circular shaped metal or plastic platters coated with magnetic material. Both the surfaces of each platter are used to store the information on the disk by recording data

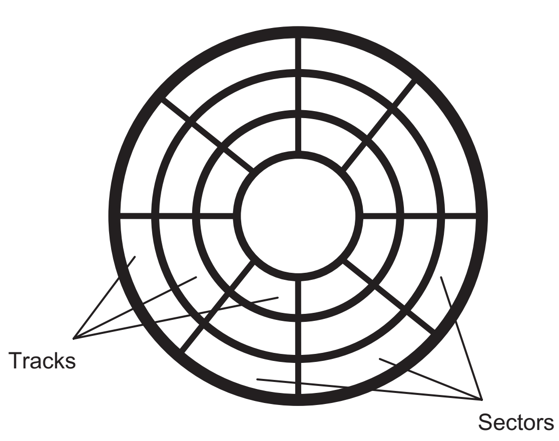


Fig. 2.10 Track and sectors on a disk platter surface

magnetically. To organize the data on the disk, each platter is logically divided in a concentric set of rings called *tracks*. There may be thousands of tracks per surface of each platter depending on the storage capacity of the disk. Each track is further divided into *sectors* (see Fig. 2.10). A sector is the smallest unit where the actual data are stored on the disk. Each track may consist of hundreds of a sectors. The size of sector may be variable or fixed. But in contemporary systems, sector size is fixed. The fixed sector size is 512 bytes. Thus, the data are transferred to and from the disk in sectors only.

The older disks store the same number of sectors per track as shown in Fig. 2.10. But, as we move outward, the number of sectors per track may be increased as there is more space on the outer tracks as compared to the inner tracks. The modern disks exploit this feature and store more number of tracks as we move from the innermost track to the outermost track. There can be various zones of tracks storing different capacities of sectors. For example Zone 1 has three tracks and each track in this zone stores 90 sectors. Similarly, Zone 2 has four tracks and each track in this zone stores 120 sectors.

The magnetic disk is mounted on a *disk drive* consisting of the following components:

Spindle

The part where the disk spins around, i.e., the spindle is a mechanism through which it rotates the disk.

Read/write head

A device that is able to read from or write to the sectors of a track on one platter surface, i.e., to transfer the information it just moves above the surface of a platter. The head may be either movable or fixed. In fixed-head disk, there is one head per track whereas in movable-head disk, there is only one head for all the tracks. The movable head moves to all the tracks. In movable-head, however, the head is separate for both surfaces of a platter, i.e., each platter has two heads.

Disk arm

The head is mounted on an arm that can be extended or retracted for positioning the head on any track. In case of a movable-head disk, there are multiple arms depending on the number of platters.

A disk in common use comes with multiple platters and movable read-write head mechanism as shown in Fig. 2.11. The tracks that appear at the same location on each platter form a *cylinder*. Whenever we need to store sequentially related information, it should be stored in a cylinder. This is done by first storing the information on a track of a platter and continuing the information on the same track of others platters.

There may be some error due to misalignment of the head or interference of magnetic fields. Due to this reason, some gap (see Fig. 2.12) is required between any two tracks (inter-track gap) and similarly between any two sectors (inter-sector gap). This gap avoids any error due to misalignment or the effect of magnetic field on adjacent tracks or sectors.

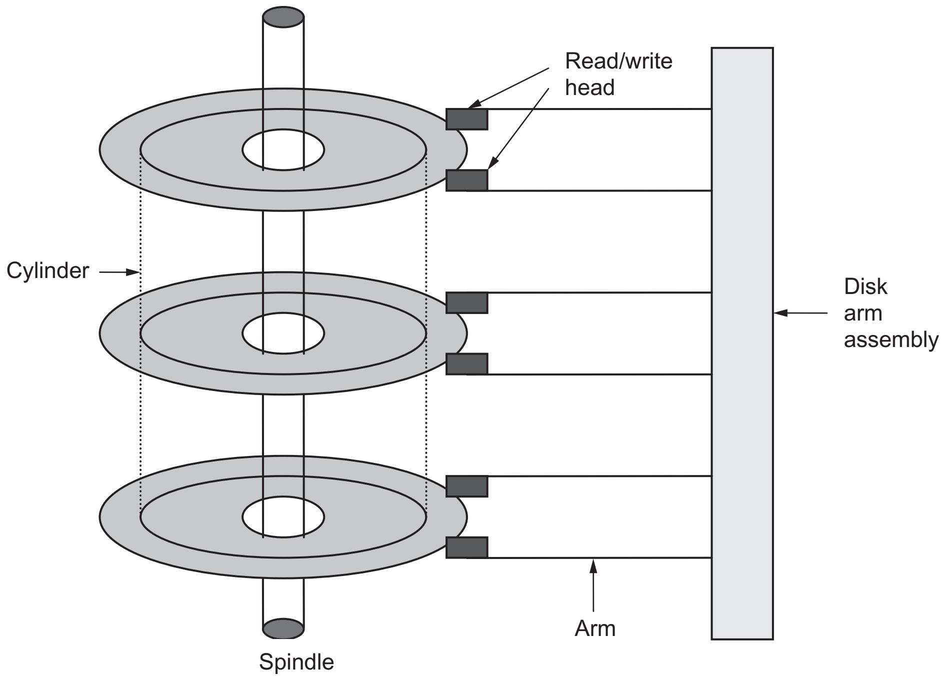


Fig. 2.11 Physical structure of a disk

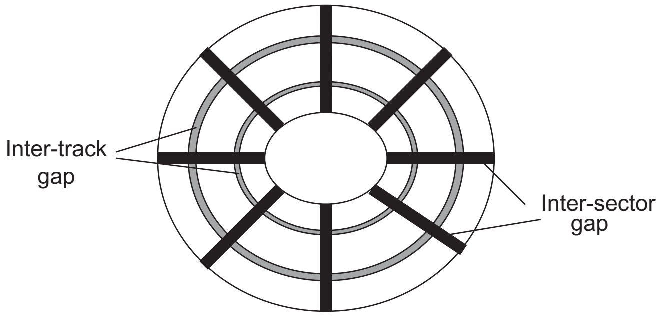


Fig. 2.12 Inter-track and inter-sector gap on the disk surface

The disk starts functioning with the help of a motor. The drive motor spins the disk at a high speed and the head performs read/write on a portion of the surface of the platter rotating beneath it. The processor initiates a disk read operation by first writing a command, then the logical block number from where the data are to be read, and finally the destination memory address to a port, which is associated with the disk controller. To find the location on the disk, the head first locates the desired track by moving on it and then the platter under the head rotates such that the desired sector comes under the head. Disk controller reads the sector and performs a DMA transfer into the memory. On completion of the DMA transfer, the controller sends the interrupt to processor for notifying it the completion of the operation.

2.10.1 Disk Formatting

The disk formatting prepares the raw disk to be used. There are three steps in disk formatting: low-level formatting, disk partitioning, and logical formatting. The *low-level formatting* is performed by the manufacturer and the other two steps are performed by the operating system and therefore are linked to it. The manufacturer of the disk performs the low-level formatting and is able to test the disk and later on use the disk for storage. The purpose of low-level

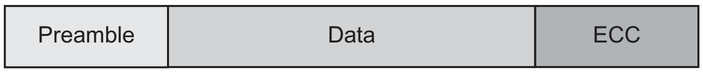


Fig. 2.13 Format of a sector

formatting is to organize the surface of each platter into entities called tracks and sectors, by polarizing the disk areas. Tracks are numbered starting from 0, and then the heads polarize concentrically the surface of the platters.The low-level format thus decides the number of concentric tracks, number of sectors on each track, sector size on the track, and the inter-track and inter-sector gaps. The format of a sector is shown in Fig. 2.13.

The preamble is a bit pattern used to recognize the start of a sector. It consists of the cylinder number, sector numbers, and other related information. The low-level format decides the size of the data field. The data size is in general 512 bytes. Error-correcting code (ECC) that contains the redundant information used to recover from read errors. The size of ECC field is decided by the manufacturer. In general it is a 16-bit field. The number of spare sectors are also reserved at the time of low-level formatting.

Thus low-level formatting reduces the actual space of a disk as some of the space is reserved for preamble, ECC, inter-track gap, inter-sector gap, and spare sectors. On the average, the disk capacity reduces by 20% after low-level formatting.

2.10.2 Disk Partitioning

On a disk, separate areas need to be created as per the convenience of the user to keep his work separate. Thus, disk partitioning is process of dividing the storage space of a hard disk into separate data areas. These separate data areas are known as partitions. For this purpose, a partition editor program may be used that creates, deletes or-modifies these partitions. After creation of different partitions, the directories and the files on various partitions may be stored. There may be two types of disks on the basis of disk partitioning: *basic disk* and *dynamic disk*.

Basic disks are the storage types most often used with Windows. The basic disk contains partitions, such as primary partitions and logical drives, and these are usually formatted with a file system to become a volume for file storage. The space can be further added to existing primary partitions and logical drives by extending them to adjacent, contiguous unallocated space on the same disk. The provision of multiple partitions on a disk appears to have separate hard disk drives to the user.

The following operations can be performed only on basic disks:

- Primary and extended partitions can be created and deleted.
- Logical drives within an extended partition can be created and deleted.
- A partition can be formated and marked as active.

The first physical sector on a basic disk contains a data structure known as the *master boot record* (MBR). The MBR contains the following:

- A boot program (up to 442 bytes in size)
- A disk signature (a unique 4-byte number)
- A partition table (up to four entries)
- An end-of-MBR marker (always 0x55AA)

Thus, there may be two types of partitions based on the above discussion: *primary* and *extended*. There may be multiple primary partitions but one of the primary partitions is used to store and boot an operating system. The primary partition that is used to boot the system is set active to indicate that this is the *boot partition*. If more than one or no primary partition is set active, the system will not boot. The extended partition is divided into logical drives and is viewed as a container for logical drives where data are located. This partition is formattable and is assigned a drive letter.

Another data structure known as the *partition table* stores the information about each partition such as its starting sector, size of each partition, etc. The partition table is also stored at sector 0 as shown in Fig. 2.14.

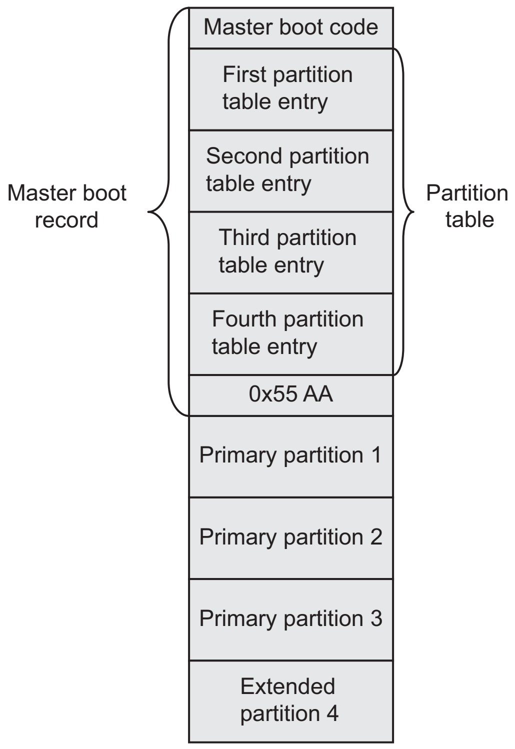


Fig. 2.14 MBR partition

Dynamic disks, on the othe hand, have the ability to create volumes that span multiple disks. The volumes thus created are known as dynamic volumes. The volume management is very flexible in case of dynamic disks as they use a database to track information about dynamic volumes on the disk and about other dynamic disks in the system. The following operations can be performed only on dynamic disks:

- Create and delete simple, striped, mirrored, and RAID-5 volumes (striped, mirrored, and RAID will be discussed in detail in Chapter 15).
- Remove a mirror from a mirrored volume or break the mirrored volume into two volumes.
- Repair mirrored or RAID-5 volumes.

Another step in disk formatting is logical formatting concerned with the operating system. This is also known as high-level format. This operation is performed for each partition. The logical formatting operation lays down a boot block in the partition and creates a file system. The initial file system data structures such as free and allocated lists or bitmaps, root directory, and empty file system are also stored. Since different file systems may be there in different partitions, the partition table entries will indicate which partition contains which file system.

SUMMARY

Let us have a quick review of the important concepts discussed in this chapter:

- Everything an operating system does is interrupt driven.
- Interrupt is a signal to the processor generated by hardware or software indicating an event that needs immediate attention.
- On the processor hardware, there is an interrupt-request (IRQ) line that the processor senses for any interrupt after each instruction execution of the process.
- There is a program known as interrupt service routine (ISR) corresponding to each interrupt generated.
- The addresses of all ISRs are placed in a list known as *interrupt vector table* (IVT).
- A hardware interrupt is generated from an external device, which could be either a part of the computer itself such as a keyboard, disk or an external peripheral.
- The software interrupts are caused either by an exceptional condition in the process, or a special instruction in the instruction set which causes an interrupt when it is executed.
- Device controller is an electronic device in the form of chip or circuit that controls the communication between the system and the I/O device.
- To communicate with each type of device controller a specific code in the form of a device driver is written that takes care of the specific device controller registers and the commands. Thus, the device drivers act as a layer that hides the differences among the device controllers.
- The modern OSs separate code and data of the OS from the code and data of the user processes. This separation is termed as *dual mode operation*. The dual mode operation has two modes: the kernel mode and the user mode.
- Initially, the mode bit is set to 0, which means the control is with the OS when the computer system is started. When a user process wants to gain the control, the mode bit is set to 1 and the user is able to execute in his own area but is prevented all access to the kernel memory space.
- The Intel modern processors come with four privilege rings (0-3).
- All I/O instructions are privileged. To access any I/O device, the process may request to the OS in the form of a system call.
- The system call is a user request to the operating system which is interpreted and executed on the hardware by the operating system on the behalf of the user.
- In programmed I/O technique, the processor time is wasted as it continually interrogates the status of I/O operation.
- In DMA-based I/O, instead of generating multiple interrupts after every character, a single interrupt is generated for a block, thereby reducing the involvement of the processor.
- There are three following steps in disk formatting: low level formatting, disk partitioning, and logical formatting. The *low-level formatting* is performed by the manufacturer and the other two steps are performed by the OS and, therefore, they are linked to it.
- The purpose of low-level disk formatting is to organize the surface of each platter into entities called tracks and sectors, by polarizing the disk areas.
- Disk partitioning is a process of dividing the storage space of a hard disk into separate data areas. These separate data areas are known as partitions.
- *Primary partition* is a partition that is required to store and boot an operating system.

MULTIPLE CHOICE QUESTIONS

1. The modern OSs are ____________.
   (a) programmed-I/O driven (c) software-driven
   (b) interrupt-driven (d) hardware-driven
2. Interrupt is a signal to the ____________ generated by hardware or software.
   (a) memory (c) processor
   (b) device controller (d) none
3. IVT is generally placed in ____________ memory.
   (a) low (c) disk
   (b) high (d) none
4. The number of hardware interrupts is limited by the number of ____________.
   (a) processes (c) IRQ lines
   (b) processors (d) none
5. ____________ is also known as an *adapter*.
   (a) memory (c) device
   (b) processor (d) device controller
6. Which of the device controller register is read-only?
   (a) control (c) data
   (b) status (d) none

7. Which of the device controller register is write-only?
   (a) control (c) data
   (b) status (d) none
8. Initially, the mode bit is set to ________.
   (a) 1 (c) 2
   (b) 0 (d) none
9. The base and limit registers are updated for every process in ________ mode.
   (a) user (c) both user and kernel
   (b) kernel (d) none
10. The first physical sector on a basic disk contains a data structure known as the ________.
   (a) partition sector (c) boot record
   (b) basic sector (d) master boot record

REVIEW QUESTIONS

1. What is an interrupt? What are its types?
2. What are the tasks to be executed when an interrupt arrives on the processor?
3. What is IVT?
4. What is ISR?
5. What is a trap?
6. Provide some examples when software interrupt is generated.
7. Provide some examples when hardware interrupt is generated.
8. How are multiple interrupts handled?
9. Differentiate between blocking and non-blocking I/O devices.
10. What is a timer? Explain its role in operating system.
11. What is a device controller? How does it work?
12. What is a device driver? Explain its functioning with device controller and operating system.
13. What were the basic problems in multi-programming-based modern operating systems?
14. What is the need of a dual mode protection?
15. What is the need of memory protection?
16. What is the need of processor protection?
17. What is the need of I/O protection?
18. Explain the physical structure of a magnetic disk.
19. What is a cylinder on a disk?
20. What is disk partitioning?
21. Differentiate between primary and extended partitions.
22. What is MBR?

BRAIN TEASERS

1. The interrupt number of an hardware interrupt is 8. At what location in the IVT, its ISR address will be found?
2. Is nested interrupt possible? If yes, how are they handled?
3. All I/O instructions are privileged. Then, how does a user access the devices?
4. Which of the following instructions should be privileged?
   (a) Switch from user mode to kernel mode
   (b) Updating base and limit register
   (c) Clear memory location
   (d) Set value of timer
   (e) Read a clock
   (f) Interrupts are disabled
   (g) Executing a loop to enter user data
   (h) Load a value in processor register
   (i) Abort a process
   (j) Read input from keyboard
   (k) Send a file to printer to print
   (l) A global variable in the user process reinitialized
5. Inter-sector and inter-track gaps are used on the disk to avoid errors. How do these gaps affect storage utilization on the disk?
6. Study the DOS and Windows operating systems with reference to dual mode protection and find out which operating system provides a better protection in terms of multi-tasking.

3 Resource Management

3.1 INTRODUCTION

The hardware resources are not easy to interface. There is a lot of complexity in using them. Moreover, there are very limited resources in the system and multiple processes. Owing to this, there should be some schedule and management for accessing and using the resources. The OS performs all these functionalities. It schedules the limited resources among multiple tasks and gives an easy interface to I/O devices. The hardware resources are abstracted or transformed into virtual devices. The virtual devices are easy to work from the user's viewpoint. The resources are of three types: hardware, virtual, and software. In this chapter, responsibilities of the OS as a resource manager are discussed, along with the types of resources and the goals of resource management. The functions and the components of the resource manager and all the components of resource management are also discussed. Since all these components are part of the operating system, all of them, along with other parts, will be discussed in different chapters of this book.

3.2 FUNCTIONS OF A RESOURCE MANAGER

The OS as a resource manager performs the following functions:

3.2.1 Resource Abstraction/Transformation

As discussed in Chapter 1, it is really difficult to work with hardware devices. To perform read or write function from I/O devices, we need to know the structure of every device in the form of registers: data registers, control registers, and so on. A user or programmer cannot work efficiently if he or she works so close to the hardware, since there are numerous details that need to be taken care of; thus, hardware resources are complex interfaces to work with. To ease the job of the user, the OS hides the complex details of the hardware and presents I/O devices to them in such a form that it is easy to interface with these devices. In fact, actual hardware devices are simulated in the form of a program known as virtual device. The user program interfaces with the virtual device, which, in turn, interfaces with the actual device. In this way, actual device has been abstracted or transformed into a virtual device and presents the user with an easy interface.

Learning Objectives

After reading this chapter, you should be able to understand:

- Operating system as a resource manager
- Transformation of hardware devices into virtual devices
- Time division multiplexing
- Space division multiplexing
- Resource scheduling
- Hardware resources
- Virtual resources
- Software resources
- Nature of resources
- Goals of resource management
- Working of resource manager
- Components of resource management

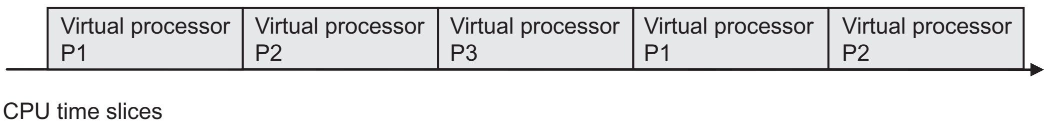


Fig. 3.1 Time division multiplexing

Besides providing an easy interface to the devices, another benefit of abstraction is that the concept of virtual devices provides the best utilization of the devices. For example, if multiple users have requested for printing, a single printer cannot handle all of them simultaneously. Moreover, it may mix up the output of many users. Therefore, multiple virtual printers can be created to give the impression to the users that they are using printers exclusively. In this way, actual single device is converted into multiple virtual devices. Another problem is that while executing the program, a fast CPU cannot cope up with the slow speed of I/O devices. It cannot wait to read from a card reader or keyboard or to print on the printer because all I/O devices are much slower as compared to the speed of a CPU. The third advantage of having virtual devices is that with the use of these devices, the program can be executed without any speed limit of I/O devices.

3.2.2 Resource Sharing/Multiplexing

As discussed in Section 3.2.1, since virtual devices will be more as compared to actual devices, there is a need to share the actual devices among the virtual devices. This is known as resource sharing or multiplexing. The resource sharing is done by following two methods:

Time Division Multiplexing

In this type of resource sharing, a device is shared by programs at different times. As seen in time-sharing systems, CPU time is shared by multiple programs. There is a single-processor, but with the help of virtual processors, single-processor time is shared. Every virtual processor is given time on the actual CPU. In this way, all virtual processes share the processor at different times. This is known as time division multiplexing or time-sharing (see Fig. 3.1).

Space Division Multiplexing

In this type of sharing, the actual device or resource is divided into smaller versions, and each virtual device is allocated a part of the resource. For example, main memory is divided into several partitions to accommodate several programs. It means that for every virtual processor, a virtual memory (VM) is needed, which is provided by dividing the actual memory (see Fig. 3.2). Similarly, hard-disk space is also divided to accommodate several programs to have the impression of separate secondary storage of their own. This is known as space-division multiplexing.

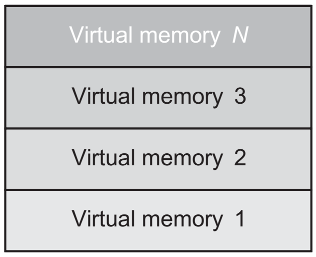


Fig. 3.2 Space division multiplexing

3.2.3 Resource Scheduling

It is a well-known fact that all the resources are limited as compared to the number of processes. That is why there is a need to schedule the processes to the limited resources. There are many instances in the lifetime of a process when scheduling is needed. Whenever a job is submitted in a batch system, it is first stored in a job pool in the hard disk. A job pool is maintained in the hard disk for all incoming jobs entering the system first time. When there is a space in the main memory, a job is brought from job queue to the ready queue in the main memory. Ready queue is the place where jobs are fetched from job pool and the jobs wait there for their turn to be executed. The process of bringing a job from job pool to the ready queue is called *job scheduling*. However, there is no need of job scheduling in a time-sharing system because jobs directly enter the ready queue instead of a job queue. Now, in the ready queue, there are multiple processes that are ready and that need to be executed on a single CPU. There is a need to schedule the processes on CPU as it can execute only one process at a time. This is known as *process scheduling*. There may be many process scheduling algorithms depending on the situation and type of the system. Similarly, limited I/O devices are needed by a number of processes. There must be a scheduling mechanism for allocation of these devices to the processes. Likewise, there are memory, virtual memory, hard disk, and files that need to be scheduled for multiple processes. As students will study more components of the OS in the book, they will come across some more scheduling concepts.

There are many schedulers needed to perform scheduling. For example, for job scheduling, there is a scheduler called long-term scheduler. Similarly, for process scheduling, short-term scheduler is used. The scheduler is the software that selects the job to be scheduled according to a particular algorithm. We will study in detail all the schedulers and their functioning.

3.3 RESOURCE TYPES

In this section, all the resources available to the OS are described. The OS needs to manage all these resources, and hence to understand the functioning of an operating system, it is imperative to learn about these resources in detail. There are three types of resources (see Fig. 3.3): hardware, virtual, and software.

Hardware resources do not require any description. The major hardware resources are processors, memory, I/O devices, and hard disk. The hardware resources that have been abstracted or transformed into other resources are known as virtual resources. The processes, virtual memory, logical devices, and files are examples of virtual resources.

The next type of resources includes virtual resources. Since the hardware resources are very complex in nature for direct usage, there is a need to hide their details such that there is some abstraction in their use that makes it easy to use and interface them. For example, when there are multiple users to share a single CPU, they cannot use it. However, if separate processes are made for each user, then these processes act as virtual processors such that the computational power of a single CPU is shared among multiple users. Similarly, the physical memory available cannot accommodate the user programs that are larger than the size of the available memory. However, with the use of VM concept using hard disk, it is possible to accommodate

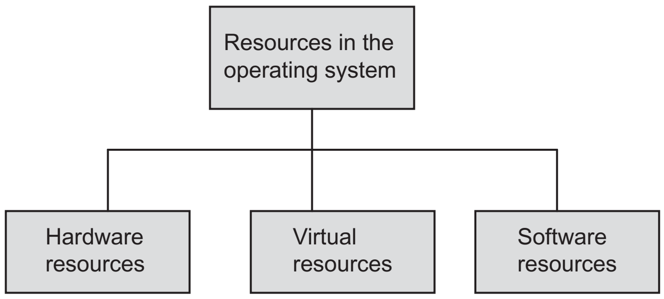


Fig. 3.3 Resource types

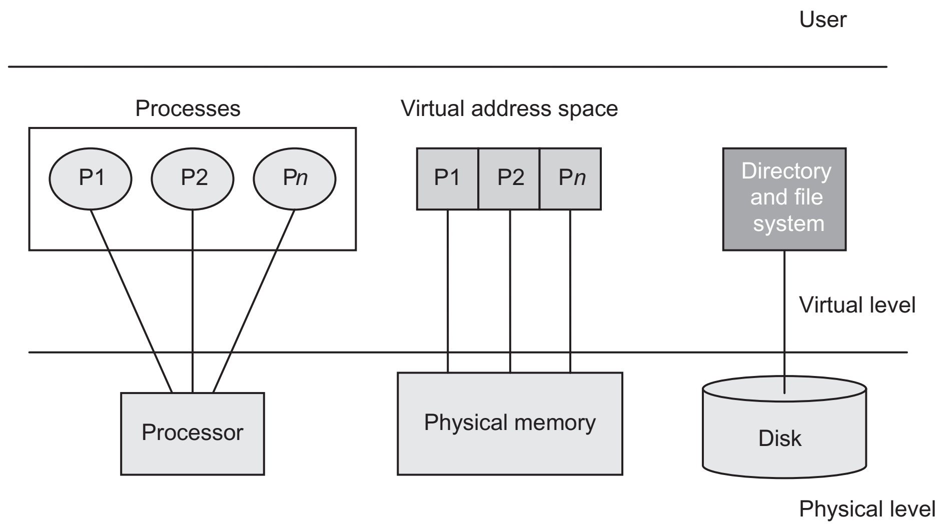


Fig. 3.4 Mapping of virtual resources to hardware Resources

the larger size programs in main memory in spite of less memory available. Similarly, hard disks are very complex if used directly. Therefore, to use them conveniently without taking care of its internal structure, the concept of files and directories are there. Files and directories are easily understandable concepts through which a user stores, updates, and retrieves his or her work on the hard disk. In this way, all hardware devices that are very complex in nature have been abstracted into virtual devices in such a manner that they do not present complex details to the user, but instead are very flexible, easy, and understandable (see Fig, 3.4).

Software resources have no direct relation with the hardware resources, that is, they are independent of hardware and virtual resources but may be used in managing them. For example, the starting address of a page (it is a logical entity used to represent the divisions of a process and is discussed in Chapter 10, Memory Management) will be stored in a page table. So the pages of a process need to obtain the slots in a page table. Therefore, page-table slot is a software resource. Similarly, for inter-process communication, messages in a message queue or mailbox are software resources. There is a resource used for synchronization between the processes known as semaphore. Semaphore is also a software resource. Other examples of software resources may be segment-table slot, file allocation table slots, local descriptor table (LDT) slots, global descriptor table (GDT) slots, and so on. The students become familiar with these resources as they progress through the chapters.

3.3.1 Nature of Resources

The resources can also be categorized according to their nature. The resources are consumable or non-consumable. Resources are also categorized based on the fact that when one resource is in use, whether it can be taken by a process or not. Based on these characteristics, the resources have been categorized as follows:

Non-consumable Resources

The resources that cannot be consumed but can be used, that is, when one process has used the resource, another process can use it. For example, memory, CPU, and I/O devices are shared by the processes. All physical resources are non-consumable resources.

Consumable Resources

The resources when used by a process are consumed and cannot be used by others. For example, semaphores, interrupts, signals, and messages when consumed by processes cannot be allocated to others.

Non-pre-emptive Resources

The resources that when allocated to a process cannot be taken away or preempted by others, that is, the process holding the resource has complete control over it. For example, when a printer is allocated to some process, it cannot be allocated to other processes until the first process finishes its job.

Pre-Emptive Resources

There are situations when resources can be preempted from the process holding it. As explained in the time division multiplexing, these types of resources need to be shared without completion of the job of a process and can be allocated to other processes. For example, CPU when allocated to a process in time-sharing environment may be preempted by another process when its time slice expires. It may be possible that within the time slice, the process holding the resource has not completed its execution but will be preempted by other processes on the expiry of its time slice. However, the process will get CPU time again when its turn comes. However, this is not easy to do. When one process is interrupted, we know that it has to be resumed when its turn comes. Therefore, the state of one process is required to be saved. Therefore, pre-emption is achieved with overhead. This will be discussed in detail in process management.

3.4 GOALS OF RESOURCE MANAGEMENT

We discussed various types of resources in Section 3.3. These resources need to be shared among the multiple processes. While allocating the resources to processes, the resource manager in the OS should take care of the following goals:

Resource Utilization

As described in the Chapter 1, all the resources must be utilized as there is always a scarcity of resources in the operating system. For example, CPU should not be idle and must be utilized. There are many concepts in the OS that originated only from this goal. As we have already discussed, multi-programming, multi-tasking, and multi-threading (discussed in detail in Chapter 9) are the concepts in response to keep the CPU busy. Similarly, VM is the concept to utilize the available physical memory.

Protection

In multi-programming and Multi-tasking environment, processes should not be allowed to access other processes' area or operating-system area, as user processes and the OS both are in the main memory.

Synchronization

Resource manager should not allow the multiple processes to access a resource that is mutually exclusive. Otherwise, there may be a chaos and the results will be disastrous. For example, if multiple processes access a shared memory location to read and write simultaneously, then there will be wrong results. It means that synchronization must be provided among processes by the resource manager.

Deadlock

When multiple processes share the resources, it may be possible that one process P1 is holding a resource R1 and waiting for another resource R2. However, R2 is held by another process P2 and P2 is waiting for R1 held by P1. In this situation, both processes are waiting for one another to release the resource. This situation is called a *deadlock*, and the processes are in a deadlocked state. It is the responsibility of the resource manager to check that deadlock condition never occurs in a system.

Fair Distribution

In some systems, such as in multiuser systems, all processes should get equal time of the CPU. Therefore, in this case, resources should be allocated to all the processes such that the processes get a fair distribution.

3.5 HOW RESOURCE MANAGER WORKS?

When a process requests for some resource, there are chances that it does not get it immediately because some other process has already acquired it. Therefore, there are queues where the process waits for its turn. There may be a scheduling criterion for processes in the queue implemented by the resource manager. The resource manager, therefore, according to the scheduling criterion, selects the process in the queue and assigns the resource as seen in Fig. 3.5. However, before assigning the resource to the process, it performs the following tasks:

Accounting of Resources

The resource manager keeps the account of number of instances of a resource to check which instance is free and which already allocated. If there are no free resources, then the process is asked to wait.

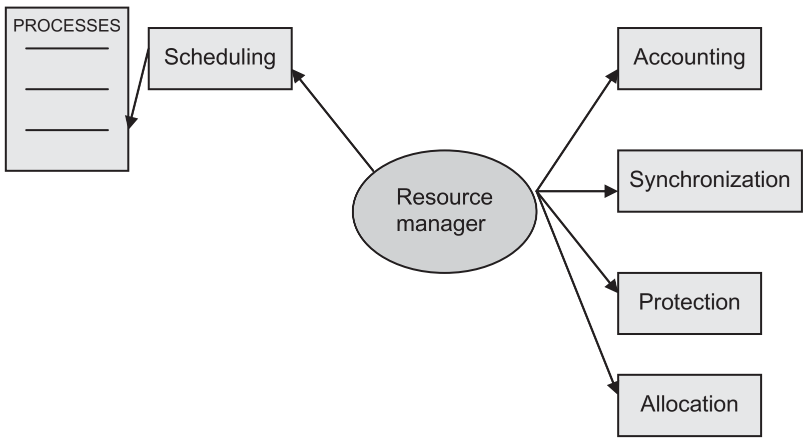


Fig. 3.5 Resource manager functions

Synchronization

If a process requests a resource, the resource manager first checks whether the resource is mutually and exclusively accessible or not. If it is not, then it cannot allocate the resource to the process and waits until it becomes free.

Protection

The resource manager should check the authorization access on a resource. If a resource is only readable, then a process should not be able to write on that resource. Moreover, if a process requests to access the memory location of the OS or other users, it should not be allowed to do so.

Scheduling

In a waiting queue, the way the processes are to be retrieved is decided by the resource manager. This process is known as scheduling.

Allocation

After passing through synchronization and protection checks, and if there is availability of resource, then the resource is finally allocated to the process.

After allocation of resources, the process uses and returns them to the resource manager so that other processes can use them.

3.6 COMPONENTS OF RESOURCE MANAGEMENT

In this section, all the components of resource manager are discussed. In other words, the way all the resources in the OS are managed and utilized is described. Largely, these components form the operating system. A brief overview of them is given here and will be discussed in detail in separate chapters throughout the book.

3.6.1 Process/task Management

The terms *job*, *process*, or *task* have been used interchangeably till now, but the meanings of all these and similar terms have been made clear in further chapters. At this stage, the students should only understand that there is a task or process to be executed by the user or the operating system. It means that there are two types of processes: the user and the OS . As discussed, multi-programming and multi-tasking introduced the concept of multiple jobs in the main memory. Therefore, there is not a single job to be executed but many. The first question is how to create and perform several operations on it. A process will have many states such as ready, executing, suspended, and terminated. All the operations performed by the user on the processes are implemented by the operating system. When a process is created, it needs certain resources such as CPU time, memory, file, I/O devices, and so on. The status of the resources and the execution of a process need to be stored somewhere in a data structure known as *process control block* (PCB). The PCB is also maintained by the operating system. It is very useful in various operations on process.

Since there are many processes to be executed in multi-programming environment, all compete for execution by the CPU. There should be some mechanism for the allocation of the CPU to one process. This is known as *process scheduling*. Process scheduling should be fair enough to all processes. The process scheduling job is performed by the operating system.

As discussed, the multiple processes have increased the problems and challenges for the operating system. When there are some shared resources, there is a need to control the access of resources by the processes such that at a time, only one process should have the control of that resource, otherwise, there may be inconsistencies. The mechanism to manage the accesses is called *process synchronization*. Process synchronization demands that the processes co-operate. For example, if two processes are sharing a data structure and if both try to access it simultaneously, there will be a data inconsistency; thus, process synchronization is a complex feature performed by the operating system. Similarly, co-operating processes need to communicate with each other. The OS provides an *inter-process communication* (IPC) mechanism by which the processes communicate. The importance of inter-process communication facility increases in a distributed environment where processes reside in geographically separate locations. The message passing system is a convenient method for this purpose.

Another problem in process management is deadlock as described in Section 3.5. The OS resolves the deadlock such that the system should be in the safe state and the normal execution of the system resumes.

To summarize, the OS performs the following process-management functions:

i) Process creation, deletion, and other operations on process
ii) Process scheduling
iii) Inter-process communication
iv) Process synchronization
v) Deadlock management

3.6.2 Memory Management

Memory management consists of two parts: main memory and virtual memory. The main memory is central to any operation done by the system. Whenever we want to execute on the system, we need to first store it in the main memory. It means that the user process should also be stored in the main memory first. The multi-programming and multi-tasking concepts require more than one process to be in the main memory. Therefore, memory must be partitioned first and allocated to the processes. The OS partitions and allocates the memory to the processes using some mechanisms. The size of memory partitions can be fixed or varied. The processes in these partitions can be allocated as contiguous or non-contiguous. The contiguous allocation means that the space allocated to a process should be contiguous in the memory. If small chunks of memory are scattered in the memory, then they cannot be allocated to a process due to their non-contiguous locations. On the other hand, non-contiguous allocation allocates the scattered memory chunks to the process in the memory. Non-contiguous allocation is implemented as *paging*. The OS keeps account of available memory partitions and occupies portions of memory. It also has an account of memory requirements of each process. In this way, keeping in view the available memory, the OS allocates the space to the process in the memory. All these concepts will be discussed in detail in the individual chapters. Based on these criteria, we have contiguous allocation and non-contiguous allocation methods. It should be noted that all memory-management mechanisms provided by the OS are supported by the hardware.

There is another concept in the memory management. When a user writes a program larger than the size of the memory, in normal memory management scheme, this program cannot be stored in the memory and executed. However, a VM concept has been invented that allows the programmers to write the programs of very large size, which, in spite of their size, can be stored and executed. The OS supports VM mechanism at some cost of secondary memory and speed. This will also be discussed further in detail.

To summarize, the OS performs the following memory management functions:

i) Keeps account of the allocated space to the processes and the available space
ii) Partitions the memory as per fixed partition or variable partition methods
iii) Allocates the memory to the processes as per contiguous or non-contiguous methods
iv) Manages VM

3.6.3 Secondary Storage Management

Since the main memory cannot accommodate every program or data, some other mechanism is required to store them. For this purpose, hard disk as a secondary storage is a widely used device for storing every program or data. We have already discussed that whenever a job enters the system, it is first entered in the job queue, which is maintained on the hard disk only. The system programs like compilers, debuggers, editors, and so on, including a large part of the operating system, are stored in the hard disk. In VM concept, we need to swap out some pages of the process from the main memory for some time. These swapped-out pages are also stored in the hard disk; thus, secondary storage provides the backup for the main memory to store programs and data. However, to utilize the hard disk for several purposes, there efficient management of space on it is required. There should be mechanisms for managing the free space on the hard disk, as it is necessary to know which part of the disk is free to be allocated to some program. Then, it must be decided how to utilize the available space in the best manner, that is, storage allocation methods should be devised. Swap space for VM should be allocated and reserved only for this purpose. Disk as a device may have queue of the processes to be accessed. Therefore, disk-scheduling techniques should be there for a better utilization of the hard disk.

To summarize, the OS performs the following secondary storage-management functions:

i) Free space management of secondary storage
ii) Allocation on secondary storage
iii) Disk scheduling
iv) Swap space management on secondary storage

3.6.4 File Management

Whenever we work on the computer system, the files are the entities where we store our work. Either we write a Word file or a C program, files are there to store the work. Basically, files are logical concepts similar to the physical files where we store or place our related work at one place. Logically, we understand files very well. We save a file after writing program into it, compile it, run it, debug it, and, later on, retrieve it. However, have you thought how these files have been implemented in the system? The operating system presents a very convenient way of representing the files to a user but implements the *file system* with the help of some physical medium such as magnetic tapes or disks. The files containing the data are stored on the physical media through the related physical devices. For example, for magnetic tapes, there are tape drives, and for disks, there are disk drives. A logical file is mapped to physical memory by the operating system, and a table is maintained for the location of each file in the storage known as *file allocation table*. There are many allocation methods to allocate space to a file on the physical medium such as hard disk. These methods are known as *file-allocation* methods. The

OS implements the allocation method that will take less space, and a file can be retrieved quickly. Before allocating space to a file on the disk, the OS must be in a position to have the account for free space on it. Therefore, the OS manages a free space list that records all free disk blocks.

When a user requests for a file operation, the OS retrieves the required file from its physical medium location and presents it to the user. The OS also provides a directory system under which related files can be arranged and stored for the convenience of the user. The directory contains information about the files, for example, its name, location, size, access rights, and so on.

A file is considered a resource of the system. Therefore, multiple processes may share the files or access the same file at the same time. It means that the protection of files is also necessary as a controlled access to the users. The OS also defines the access rights to the users for a file such as read, write, execute, and so on.

To summarize, the OS performs the following memory-management functions:

i) File operations such as creation, deletion, and so on.
ii) Directory creation and deletion
iii) File-allocation methods
iv) File-retrieval methods
v) Free-space management
vi) File protection

File management will be discussed in detail in Chapter 13.

3.6.5 Input–Output Management

The most challenging task for an OS is to manage the I/O devices in a computer system. It acts as an interface between devices and other computer systems. This interface should be simple, easy to use for a user, and preferably same for any type of device. However, today, there are myriad I/O devices. Each I/O device has its own detail and complexity. In this case, the OS needs to be changed to incorporate every newly introduced device. Therefore, the I/O functionalities should be treated separately in the OS so that the other parts of the OS are not affected. The software that deals with the I/O is known as *I/O software* or *I/O subsystem*. In I/O software, there are two types of modules. First module deals with the general functionalities when interfacing with any type of device, that is, these functions are common while interfacing with any I/O device and are known as *device-independent I/O software*. For example, there should be a general interface for any type of device. The second module provides device-specific code for controlling it and is known as *device driver*. The second module in fact takes care of the peculiarity and details of a particular device, that is, how to read or write data to the device. In this way, the OS needs not to change its code again and again to incorporate any new device. Its I/O software takes care of all the I/O devices to be interfaced with the system without changing the code of it. I/O management will be discussed in detail in Chapter 14.

3.6.6 Security and Protection

In this age, various types of confidential information are being stored either on computer systems or transmitted over the Internet. However, the information/data are not safe from the security breaches. The OS must be able to secure the computer system from outside attacks. In general, it is done by providing passwords. Moreover, as a resource manager, the OS should also protect the resources either from the inside users or outside hackers if they are successful in entering the system. Therefore, the OS must prohibit the processes or users from accessing the resources if they are not authorized for them. Each object, either hardware (CPU, memory, disk, printers, etc.) or software (processes, files, databases, etc.), has a set of operations that can be performed on it. These set of permitted operations corresponding to an object are known as *rights*. These rights can be viewed as a kind of permission to access an object. When a process or user tries to access an object, its rights must be checked first. If the user tries to access the resources he or she has been authorized, then only the access will be granted, otherwise, it is denied. The pair of object and its rights is known as a *domain*. The OS provides the protection in the form of these domains. Security and management will be discussed in detail in Chapters 16 and 17.

SUMMARY

Operating systems primarily are resource managers. The hardware resources such as the CPU, memory, I/O devices, secondary storage devices, and so on are managed only by the OS, and this management includes not only using the resources but also utilizing them properly. As a resource manager, the OS also hides the unnecessary details of the hardware resources from the user and abstracts the resources in such a manner that the user does not worry about the configuration of the hardware resources. For example, the user only knows the process but not the CPU. Of course, the process will be executed on the CPU, but the user is not aware of the execution details of its process, that is, when the process will be sent to the CPU, how the process has been scheduled for how much time. These issues are not concerned with the user. Therefore, the OS abstracts the hardware resources into virtual resources. Besides the hardware and virtual resources, there are also some software resources that again need to be managed by the operating system. A message in message queue to be consumed by a process is a software resource; thus, the OS manages three types of resources: hardware, virtual, and software. Moreover, the resources can be classified based on their nature. Some resources can be consumed and preempted. Based on this, the resources can be non-consumable, consumable, preemptive, and non-preemptive.

The management of these resources is not an easy job. In a multi-programming and multi-tasking environment, the resources need to be shared by the processes, and in case of non-consumable resources, there is a need to keep account of which resource is free and which resource has been allocated. Further, in case of pre-emptive resources, when the process is preempted its state, it must be saved so that it can be resumed again. There are many issues regarding the resource management. In this chapter, all the resource types, resource-management goals, working of resource manager, including the components of resource manager, have been discussed.

Let us have a quick review of the important concepts in this chapter:

- There are three types of resources: hardware, virtual, and software.
- The hardware resources that have been abstracted or transformed into other resources are known as virtual resources. The processes, virtual memory, logical devices, and files are examples of virtual resources.
- Software resources are the resources that have no direct relation with the hardware resources. It means that they are independent of hardware and virtual resources but may be used in managing them. For example, messages in a message queue or mailbox are software resources.
- The OS abstracts the hardware devices into virtual devices.
- The OS allows the resource sharing by two methods: time division multiplexing and space division multiplexing.
- Time division multiplexing means to share the mutual exclusive resource at different times by the processes, for example, the CPU.
- Space division multiplexing means to share the resource at the same time. For example, memory can be partitioned and allocated to different processes at the same time.
- The OS schedules the resources according to a scheduling algorithm, that is, a criterion by which the resource is shared. For example, the CPU is scheduled among different processes according to first come first served criterion.
- The main goals of resource management are resource utilization, protection, synchronization, deadlock prevention, and fair distribution.
- The resource manager is responsible for accounting of resources, synchronization among processes, protection of processes, scheduling of processes, and allocation of resources.
- The resources can be classified based on their nature. Some resources can be consumed and preempted. Based on this, the resources can be non-consumable, consumable, pre-emptive, and non-pre-emptive.
- Main components of resource management are process/task management, memory management, secondary storage management, file management, I/O management, and security and protection.

MULTIPLE CHOICE QUESTIONS

1. The processes, VM, logical devices, and files are examples of
   (a) hardware resources (c) software resources
   (b) virtual resources (d) none
2. ____are the resources that have no direct relation with the hardware resources.
   (a) hardware resources (c) software resources
   (b) virtual resources (d) none
3. All physical resources are _____ resources.
   (a) non-consumable (c) pre-emptive
   (b) consumable (d) none
4. Printer is a ______resource.
   (a) non-pre-emptive (c) pre-emptive
   (b) consumable (d) none
5. The CPU is a ______resource.
   (a) non-pre-emptive (c) pre-emptive
   (b) consumable (d) none
6. The status of the resources and the execution of a process need to be stored somewhere in a data structure known as
   (a) status block (c) PCB
   (b) resource block (d) none
7. Non-contiguous memory allocation is called
   (a) memory partitions (c) demand paging
   (b) paging (d) none
8. Which of the following is a memory-management function performed by the operating system:
   i) Keeps account of allocated space to the processes and available space
   ii) Partitions the memory as per fixed partition or variable partition methods
   iii) Allocates the memory to the processes as per contiguous or non-contiguous methods
   (a) i and ii (c) i and iii only
   (b) i only (d) all
9. Swap space for VM should be allocated and reserved in ______.
   (a) main memory (c) ROM
   (b) logical memory (d) disk
10. The OS provides a ______ system under which related files can be arranged and stored for the convenience of the user.
   (a) file (c) directory
   (b) disk (d) none

REVIEW QUESTIONS

1. What is resource abstraction? Explain with an example.
2. What are the benefits of resource abstraction?
3. What is the difference between time division multiplexing and space division multiplexing?
4. Give examples of hardware, software, and virtual resources.
5. How are hardware resources mapped into their virtual resources?
6. Explain various resource-management functions.
7. What is the difference between (a) consumable and non-consumable resources and (b) pre-emptive and nonpre-emptive resources
8. Give examples of the resources asked in Question 7.
9. Name the resource type of the following: process, semaphore, memory, VM, file, and page table
10. What is a deadlock?
11. What is process synchronization?
12. What are the methods to allocate a process in memory?
13. What is VM?
14. What is free-space management?
15. What is disk scheduling?
16. What is swap space management?
17. What is a file system?
18. What is the purpose of file allocation table?
19. What is I/O subsystem?
20. What is a device driver?

BRAIN TEASERS

1. Is it true that the role of IPC mechanisms will increase in real-time systems?
2. What is the cost incurred in resource abstraction?
3. Is it possible to implement time division multiplexing on a system with multiple processors?
4. If VM is not there, what will be the effect on performance of the system?

4 Operating System Architectures

4.1 INTRODUCTION

In this chapter, the basic working of an OS and related terminologies is presented. Booting is the start process of an OS through which it sets up its environment and starts working. After booting, the OS begins its initial process and hands over the control to the user process. Since a user is not allowed to perform any I/O operation, all these are performed by the OS. However, a user needs to request the OS for all I/O operations. System call is the medium through which a user sends the request. The OS works on the hardware on behalf of the user and provides the results back to the user. Thus, system call execution and its types are fundamental for understanding the working of the OS. After a discussion of the details of the working, various architectures developed have been discussed in this chapter. The architectures have been evolved over time, catering to the needs of various requirements and keeping pace with technological advancement in computer hardware.

4.2 GENERAL WORKING OF AN OPERATING SYSTEM

The role of an OS starts as soon as the computer system is switched on and lasts till it is shut down. But where is the OS in the computer system? How does the OS get loaded in the system? How does it start? These are some questions addressed here in this section. Before delving into the working of an OS, there are some basic definitions/concepts that need to be understood.

Learning Objectives

After reading this chapter, you should be able to understand:

- General working of an OS
- Booting of the OS
- System calls, their execution, and types
- System programs
- System generation programs
- General structure of OS
- Monolithic architecture
- Layered architecture
- Virtual machine OS
- Microkernel architecture
- Exokernel architecture
- Hybrid architecture
- System generation

4.2.1 BIOS

Basic Input-output System (BIOS) is a software that basically consists of input-output functions. These functions are low-level routines that the OS uses to interface with different I/O devices, such as keyboard, mouse, monitor, ports, and so on. This is the reason that this software is named as such (BIOS). However, the meaning of BIOS was extended beyond this functioning. Since the OS is on the hard disk, it needs to be loaded onto the main memory to start the functioning of the system. So the problem is to find a way to tell the microprocessor to load the OS. It needs some instructions that, when executed, load the OS. These instructions are provided by the BIOS. In this way, along with providing the basic input-output low-level routines, it also provides the initialization function. This is the reason that the BIOS is embedded in the ROM or flash-RAM so that whenever the system is switched on, the initial instructions get executed automatically and the process of loading the OS initiates. However, it was found that BIOS was inefficient for OSs such as Linux and Windows written for 32-bit CPUs. Therefore, with the advent of new CPU architecture and development in OSs, BIOS is getting replaced. For example, since 2008 in x86 Windows systems, Extensible Firmware Interface (EFI) booting is being supported. The EFI is more flexible in accessing devices. In this way, today, BIOS is primarily used for loading the OS and initialization purposes and otherwise not used, as during the general operation of the system. Instead of calling BIOS, OSs use device drivers for accessing the hardware.

4.2.2 Booting/Bootstrapping

When a system is switched on the first time, it does not have an OS. We need to get the OS ready from the hard disk or other secondary storage onto the memory. A set of sequence of operations is needed to load the OS. This process of placing the OS in memory is known as *booting* or *bootstrapping*.

4.2.3 Boot Software/Boot Loader/Bootstrap Loader

The set of instructions needed for booting, that is, to load the OS in RAM is known as *boot software/boot loader/bootstrap loader*.

4.2.4 Boot Device

The OS is originally stored in a non-volatile secondary storage such as hard disk, CD, and the like. In the process of booting, there is a need to search this storage device, where an OS is stored, to load it onto the RAM. The device that stores the OS is called *boot device*.

4.2.5 Privileged Instructions

There are some operations, provided in the form of instructions, that need to interact with hardware devices. But a user is not allowed to access the devices directly. The instructions are first passed on to the OS, and the OS then interacts with devices on behalf of the user. Thus, the instructions, which are not directly executed by the user but need to be passed to the OS, are known as *privileged instructions*.

4.2.6 System Call

All privileged instructions, that is, instructions, which need to interact with hardware and other resources, and therefore passed on to the OS for execution, are known as *system calls*. For example, when the user needs to write something on the screen, he/she writes the output instruction in the appropriate format. This instruction in the program is system call.

The general working of an OS is discussed in the following steps:

Initialization

It was discussed that the OS acts as a resource manager, so it must have the initialized set of devices in the system. Therefore, whenever the computer is switched on, the control is transferred to the BIOS in the ROM by the hardware. The first job for the BIOS is to initialize and identify system devices such as the video display card, keyboard and mouse, hard disk, CD/DVD drive, and other hardware. This initialization job is known as *power on self test* (POST). It is a built-in diagnostic program that initializes and configures a processor and then checks the hardware to ensure that every connected device is present and functioning properly. In other words, it tests the computer to make sure it meets the necessary system requirements and that all the hardware is working properly before starting of the system. There may be some errors while the execution of POST. These errors are stored or reported through auditory or visual means, for example, through a series of beeps, flashing LEDs, or text on a display.

Booting

(a) After the execution of POST, the BIOS determines the boot device, for example, floppy, CD, or hard disk.
(b) BIOS contains a program that loads the first sector of the boot device called *boot sector*.
(c) The boot sector contains a program. The program in boot sector, when loaded onto the memory and executed, first examines the partition table at the end of the boot sector to determine which partition is active.
(d) In the partition, there is a boot loader/bootstrap loader, which is now loaded onto the memory by the boot sector program (see Fig. 4.1). The area where the boot program/loader is stored is called *boot block* of the boot device.
(e) Boot loader contains the instructions that, when executed, load the OS onto the main memory (bootstrapping) and the control is passed on to the OS by setting bits, corresponding to the privileged mode. It means that whenever the system boots, the control is with the OS, that is, the CPU is in privileged mode.

Start the Operation

(a) After being loaded and executed, the OS first queries the BIOS to get the configuration information.
(b) For each device, it checks the corresponding device driver. After confirming all the device drivers, it loads them into the kernel.
(c) The OS initializes its tables, creates needed background processes, and kicks off the start-up of the first process, such as the login program.
(d) The user programs are loaded onto the memory as the users log in. The control is transferred to the scheduler that selects a user program from the memory and transfers control to the selected program by setting bits corresponding to the user mode, that is, the CPU is now in user mode.

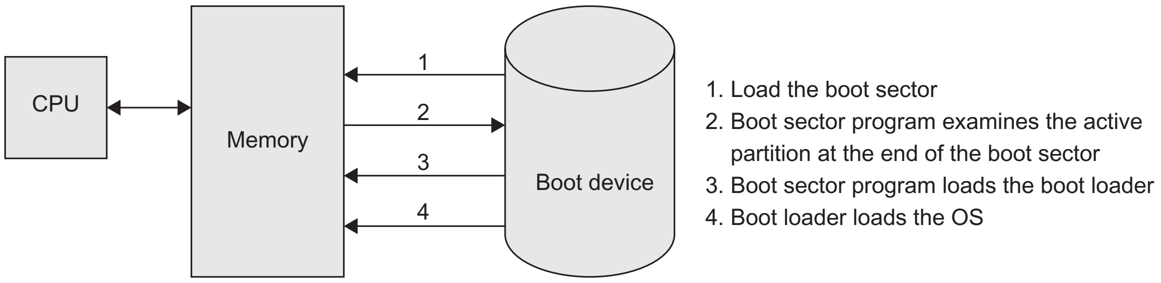

CPU
Memory
1
2
3
4
Boot device
1. Load the boot sector
2. Boot sector program examines the active partition at the end of the boot sector
3. Boot sector program loads the boot loader
4. Boot loader loads the OS

Fig. 4.1 Booting sequence

(e) Given the interrupt-driven nature of the OS, it waits for an event. When there is an event signalled by the interrupt from the hardware or software, it starts responding. The hardware interrupts are triggered by sending a signal to the CPU on the system bus. Software interrupts are triggered by executing a special operation or control instruction called a *system call*. For instance, when a user wants to access an I/O device, he/she will send a request to the OS in the form of a system call, which, in turn, executes a software interrupt and transfers the control to the OS.

(f) These system calls are handled by the system call handler that identifies the cause of interrupt by analyzing the interrupt code and transfers control to the corresponding interrupt-handling routine/event handler in the OS. For example, if there is an I/O interrupt, then control is passed on to the I/O interrupt handler. Thus, the OS has many event handlers that are invoked through the system calls.

4.3 SYSTEM CALLS

The role of system calls is important for understanding the operation of the OS. It is clear now that there are two modes in the operation of the system, that is, user mode and system mode. In the user mode, all user processes are executed and in system mode, all privileged operations are executed. The user programs and kernel functions are being executed in their respective spaces allotted in the main memory partitions. But it is obvious that user mode programs need to execute some privileged operations, which are not permitted in the user mode but allowed in the system mode. Since the processor prevents direct access to kernel-mode functions, user-mode programs must use an interface, which forms the only permitted interface between user mode and kernel mode. This interface is called system call. It means that the system call is an interface between the user programs and the OS. System calls expose all kernel functionalities that user-mode programs require. In other words, system call is an instruction that requests the OS to perform the desired operation that needs hardware access or other privileged operations. Whenever the user uses privileged instructions in the program, he/she uses system calls. For instance, when the user wants access to some hardware operations or resources like files and directories or communication with other processes, he/she uses system calls.

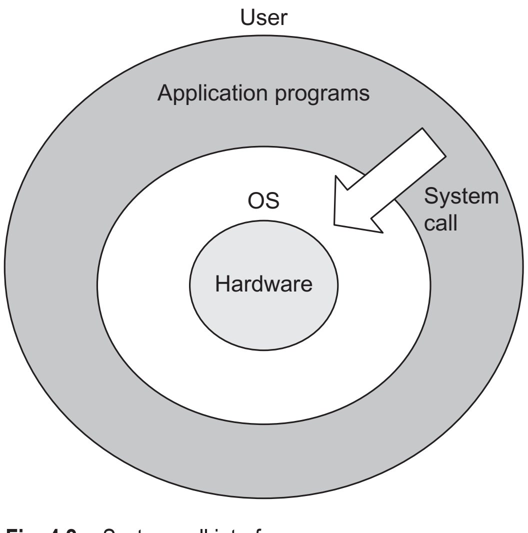


Fig. 4.2 System call interface

But it should be clear here that a system call does not perform the operations itself. System call, in fact, generates an interrupt that causes the OS to gain control of the CPU. The OS then finds out the type of the system call and the corresponding interrupt-handler routine is executed to perform the operations desired by the user through the system call. Therefore, system call is just a bridge between user programs and the OS for executing the privileged operations as shown in Fig. 4.2.

System calls are inherently used for security reasons. Due to the use of system calls, a user program is not able to enter into the OS or any other user's region. Similarly, I/O devices are also safe from any misuse by the user. Thus, through the use of system

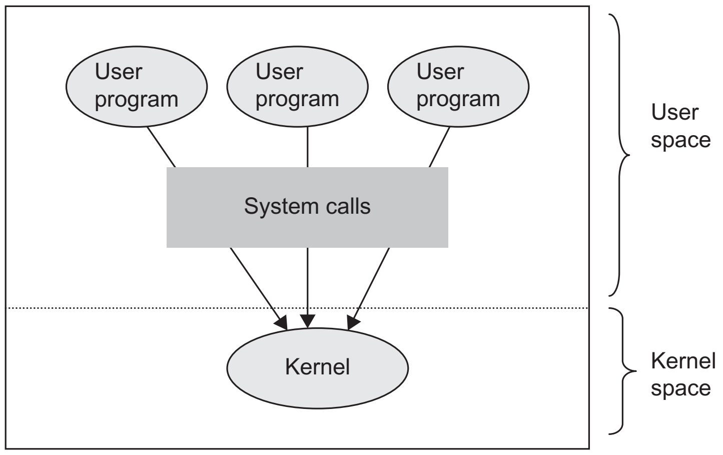


Fig. 4.3 Kernel space and user space in main memory

calls, kernel, other user programs, and I/O devices are safe and secure from malicious user programs (see Fig. 4.3).

4.3.1 Making a System Call

It is obvious that system calls for executing privileged operations or system operations, are used in a process while writing its code. System calls are generally available in the form of assembly language instructions. But with the introduction of system programming in high level languages like C or C++, system calls are directly available and used in high level languages. Therefore, it has become easy to use system calls in a program. For a programmer, system calls are same as calling a procedure or function. System calls demand nothing extra and the format is similar to that of a normal function call. The only issue is that these system calls should be available in the form of a library. The difference between a system call and a normal function call is that a system call enters the kernel but a normal function call does not. Another difference is that a system call itself does not execute anything but generates an interrupt that changes the mode to system mode and passes control on to the OS.

4.3.2 Executing the System Call

As discussed earlier, a system call is not a general function call. There is a sequence of steps to execute a system call. For execution of system call, there is the need to pass various parameters of system call to the OS. For passing these parameters to the OS, three methods are used, as follows:

1. Register method, wherein the parameters are stored in registers of the CPU.
2. If parameters are more in number, compared to the size of registers, a block of memory is used and the address of that block is stored in the register.
3. Stack method, wherein parameters are pushed onto the stack and popped off by the OS.

Another parameter to be passed on to the OS is the code of the system call being used in the user process. There is a code or system call number that is to be placed in the processor register. This is, in fact, performed by the mechanism of a system call table. A table consisting of system calls and their numbers are maintained. The numbers may differ according to different processors and OSs. When a user uses a system call in his/her program, the number of that system call is retrieved from the system call table and placed in the processor register. However, the user or programmer who writes the program does not need to worry about all these system call numbers. The library function being called in response to the system call retrieves the system call number and places the same in the processor register. The set of library functions are included with the compiler of the high-level language that makes a run-time support package.

The kernel retrieves the system call number of the system call and needs to execute the corresponding system call handler. For this purpose, it uses the system call dispatch table that stores the system call number and the location of its system call handler. After finding the address, it dispatches to execute the handler.

The following is the sequence in which a system call is executed (Fig. 4.4):

1. In the user program when the system call is executed, first of all, its parameters are pushed onto the stack and later on saved in the processor registers.
2. The corresponding library procedure for the system call is executed.
3. There is a particular code for every system call by which the kernel identifies which system call function or handler needs to be executed. Therefore, library procedure places the system call number in the processor register.
4. Then the library procedure traps to the kernel by executing interrupt instruction. With this interrupt execution, the user mode switches to kernel mode by loading Program Status Word (PSW) register to 0.

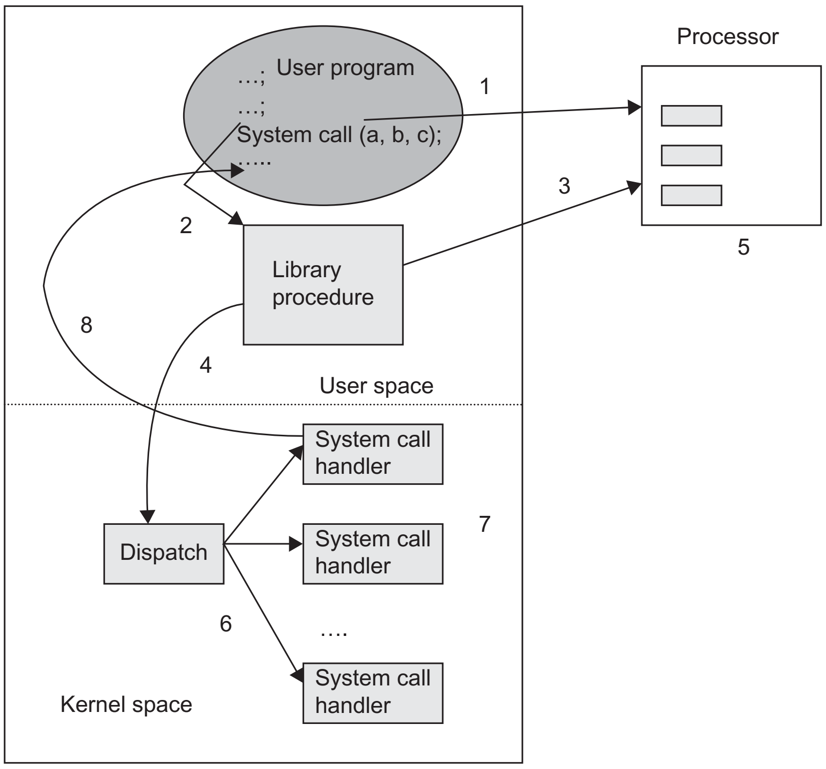


Fig. 4.4 Steps to execute a system call

5. The hardware saves the current contents of CPU registers, so that after executing the system call, the execution of the rest of the program can be resumed.
6. The kernel identifies the system call by examining its number and dispatches the control to the corresponding system call handler.
7. The system call handler executes.
8. On completion of system call handler, the control is returned to the user program and it resumes its execution.

4.3.3 Types of System Calls

Since a user needs to access many resources, the type of system calls depends on the use of these resources. For example, the user needs to have system calls related to process control and management, such as creating a process, putting a process in wait, exiting a process. Similarly, for file management, we must have system calls such as creating a file, deleting a file, opening and closing a file, reading a file, writing to a file, and so on. In this section, let us have a look at some important system calls. Basically, there are five broad categories of system calls.

Process Control System Calls

As discussed in Chapter 3, a process is a basic entity in the system. The processes in the system need to be created, deleted, and aborted. Besides these, many operations are required on the processes for their management. All these operations are performed by system calls (see Table 4.1).

File Management System Calls

A file is also a basic resource accessed by a user. Every work the user performs and stores is as files. Therefore, there is a need to access this resource through system calls. Creation, deletion, opening, closing, reading, and writing are some of the general operations on files. Similarly, for organizing files, there is a directory system and thereby system calls for managing them (Table 4.2).

Table 4.1 Process control system calls

System call	UNIX example
Create a process: Creating a new process	`fork()`
Terminate a process: When a process executes its operation, it exits normally.	`exit()`
Terminate a process abnormally: There may be situations in which you need to terminate the process in between; for example, there is hang situation, program has been stuck in an indefinite loop, and the performance of system has been affected such that no processing is being performed.	`kill()`
Increase the priority of a process: Some processes have got more importance than others. So their execution must get priority over others. This is done by setting and increasing the priority of the process.	`Nice()`
Suspend the process: There may be situations in which a process needs to be suspended but not terminated. It will resume again after receiving some signal.	`pause()`
Cause the process to sleep: A process may need to wait for I/O devices. In that period of time, the processor switches to some other process and the current process is blocked to wait or sleep for some time.	`wait()`

Table 4.2 File management system calls

System call	UNIX example
Create a file: Creating a new file.	`Creat()`
Open a file: Opening a file that is already created.	`Open()`
Close a file: Closing a file that has been opened earlier.	`Close()`
Read a file: Reading a file that has been opened earlier.	`Read()`
Write a file: Writing into a file that has already been opened.	`Write()`
Change the position of the read-write pointer: There is a need to access any part of a file randomly. File pointer indicates the current position in the file. This call changes its position as desired.	`Lseek()`
Give another name to a file: This call allows a file to appear in different directories with different names. But the copy of the file is single. It means that if there is change in the file, it is visible in all the directories wherever it appears. This is done through the unique ID of the file (known as i-number), which is an index into a table of entries known as *i-nodes*. These entries in the table store the information of a file, such as who owns the file, its disk blocks, etc. So, the i-number is same for all entries of the file in different directories. However, there is a single file; only the name is different under different directories. The file is accessible through either name.	`Link()`
Delete a file in a directory: This call removes the entry of a file in one directory.	`Unlink()`
Make a directory: Create a new directory.	`Mkdir()`
Remove a directory: Delete an existing directory.	`Rmdir()`
Change the directory: When you are working in a directory, you can move to some other directory by using this call.	`Chdir()`
Change the mode: There are various modes and groups of users who will use the files. For a particular group, there may be different access permissions (modes) such as read, write, or execute. This call changes the access permissions of a file to the specified mode.	`Chmod()`
Change ownership of file: Changes the owner and group of the indicated file.	`Chown()`

Device Management System Calls

The user cannot work without accessing the I/O devices. However, accessing them directly is not possible. Therefore, system calls are there for accessing the devices. The general commands related to this category are request of the device, release of the device, read and write operations, and so on. Since files are treated as virtual devices, most of the system calls related to file systems are used for device access also.

Information Maintenance System Calls

Some of the system calls are for accounting and providing information to the user. This information can be about a process, memory, device, computer system, OS, disk space, and so on (Table 4.3).

Communications System Calls

There is a need for communication among the processes in the system. All communication operations are also performed through system calls. The general operations in this category are opening and closing the connection, sending and receiving messages, reading and

Table 4.3 Information maintenance system calls

System call	UNIX example
Get process identification number: Every process has a unique identification number. If the user wants to see the same, this call is used.	`Getpid()`
Get status information of a file: The information regarding a file such as the file type and its permissions, size, time of last access, and so on.	`Stat()`
Set the system date and time	`Stime()`
Get statistics about a file system: Gets the statistics about a file system such as number of free blocks, file system name, and so on.	`Ustat()`

writing messages, and so on. These system calls may be related to the communication between processes either on the same machine or between processes on different nodes of a network. Thus, inter-process communication is provided by the OS through these communication-related system calls (Table 4.4).

Table 4.4 Communications system calls

System call	UNIX example
Sending a message	`Msgsnd()`
Receiving a message	`Msgrcv()`

4.4 SYSTEM PROGRAMS

These are some utilities programs above the layer of the OS, that is, programs that help a user in developing and executing his/her applications. System programs should not be confused with system calls. System programs are utilities programs that help the user and may call further system calls. For example, creating a file is a system program, which helps in creating a file, and this system program calls the system call for doing this. Thus, system call and system programs are not the same. Some examples of system programs are: file management programs (create, delete, copy, print, and so on), compilers, assemblers, debuggers, interpreters, loaders, editors, communication programs (establishing connections, sending email, browsing web pages, and so on), and status information programs (date, time, disk space, memory, and so on).

4.5 SYSTEM GENERATION PROGRAMS

Although the general architectures of a computer system and OS are the same for all machines, there may be some differences of configuration. For example, machines may differ in processor speed, memory size, disk size, I/O devices available in the system, and so on. Therefore, the OS must be configured according to the specifications available on the system on which it has to run. For this purpose, the detailed description of the configuration of the machine is stored on a file, or the hardware is directly probed at the time of booting. The description of the configuration may be in terms of the following:

- The processor type and its options selected
- Disk formatting information, its partitions
- Size of memory
- CPU scheduling algorithm

- Disk scheduling algorithm
- I/O device type and model, its interrupt number

The system generation program takes the input from the file of description about the configuration details and generates the OS accordingly. In system generation, it basically selects some code modules from the system generation library as per the hardware details and compiles and links these modules to form the OS.

4.6 GENERAL STRUCTURE OF OS

It is clear now that the OS resides in the main memory. However, as the size of the OS increased, there was the problem of keeping it in the limited memory. Therefore, the OS was structured into two parts: *Resident part* or *kernel* and *Transient part* (see Fig. 4.5). The resident part contains programs that are crucial and needed always. Therefore, they must reside in the memory forever. Thus, this makes up the resident part. The transient part is based on programs that are not always needed and, therefore, need not be in the memory forever. Therefore, this part is loaded only when needed, thereby reducing the memory requirement. In this way, the resident- and transient-part programs structure the OS.

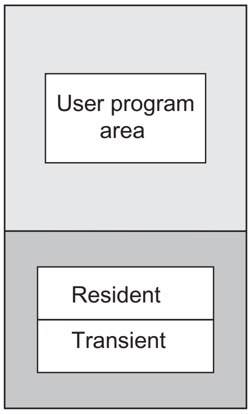


Fig. 4.5 Resident and transient parts of an OS in the memory

The decision to put programs or data structures in resident or transient part depends on its frequency of use in the OS. If the function is inherent and used every time in the operation of the OS, it must be included in the resident part. Otherwise, it should be in the transient part.

The resident part may contain the following programs or data structures:

Resource Allocation Data Structures

The data structures related to the allocation of various resources must be in the resident part. A device, if allocated to some process, needs to be checked before being allocated to some other process. Therefore, this data structure is important for smooth allocation of resources.

Information Regarding Processes and their States

Every process contains some information such as its ID, priority, state, and so on. This information is useful while switching from one process to another. The switching operation is quite frequent in multi-programming OSs. Thus, this data structure should also be in the resident part.

System Call Handler

Since the user uses system calls frequently in his/her programs, there is a need to handle these system calls by the OS. Therefore, system call handler must be in the resident part.

Event Handlers

Some event handlers are also necessary for the working of the OS, that is, they are in frequent use. For example, the memory handler or disk I/O handler is required for almost every operation.

Scheduler

This is a frequently used program in the OS whose job is to select the jobs in the queue and send it for dispatching. As we know, in a multi-programming system, there always are jobs to be executed in the queue. Therefore, there is a need to schedule them as well. So, this program should also be in the resident part.

4.7 MONOLITHIC ARCHITECTURE

In the evolution of the OS, it was demonstrated that it was developed in response to the various requirements realized. Keeping the CPU busy, multiple jobs in batch system, multiple jobs in multi-user environment expecting immediate response, user friendliness, and so on, were some of the motivation points against which OSs were designed. In this journey of OS development, one can easily realize that the development was not planned and the initial architecture of OSs was not efficient. The OSs were developed in the same way as in programming, where we keep on developing the program in one file or adding some functions and calling each other without any boundary between them. Initially, the OS consisted of very few modules, due to limited functionality. Therefore, all the functionalities were added in the kernel only. The advantage of this type of architecture was that intercommunication between the modules of the OS was efficient, as all the modules were in the kernel together (Fig. 4.6).

Later on, due to multi-programming and its extended concepts, the size of the OS grew beyond limit. This resulted in a complex structure of OS because every module in the OS accessed hardware directly. Thus, programming effort was high because there is a large gap between the meaning of operations required by the user and the meaning of operations performed by the hardware. This gap is known as *semantic gap* between the user application and bare hardware. When a user creates a process, one process or task is being created for the user. For the OS, it is a collection of some algorithms like allocating the memory for the process, scheduling the process, and so on. But at the hardware level, these operations are performed at the level of machine instructions. Therefore, there is a large gap in understanding the operations at OS level and machine level as shown in Fig. 4.7.

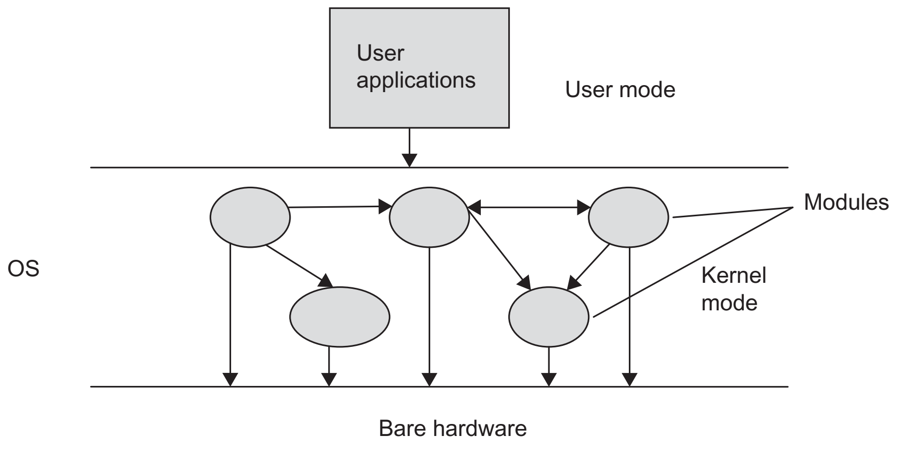


Fig. 4.6 Monolithic architecture

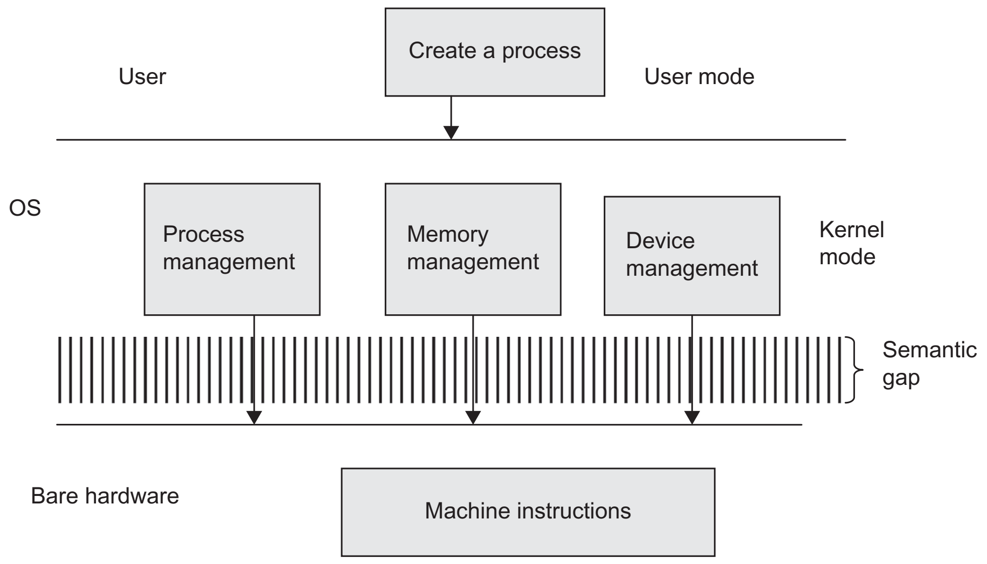


Fig. 4.7 Semantic gap between the OS and hardware

Due to all functionalities merged in a single layer, it was difficult to do modifications in a module. This is because, as a result of a lot of interfacing among modules, it is hard to imagine which module may be affected due to a single change in a module. Consequently, debugging in the modules of the OS became a difficult job. Another disadvantage of this architecture was that there was no protection. Since in this structure, there is unrestricted access of any module to the system and among them, therefore, there was the danger of malicious code. A user job can enter into the area of another job or even of the OS.

Monolithic systems were not suitable for multi-programming/multi-tasking environments due to unprotected behaviour of the system. Any user's job can corrupt any other user's job and even OS. For example, in DOS, any user job has direct access to BIOS device drivers. Therefore, there it is possible to change the functionality of any device driver. The DOS, initial architecture of UNIX, and Linux are some examples of monolithic structures.

4.8 LAYERED ARCHITECTURE

With the advancement in OSs, monolithic structures became complex and then obsolete after some time. There was the need to design the OS such that there is no single layer consisting of all the functionalities. This resulted in layered architectures (see Fig. 4.8). This architecture is based on the following two points of design:

4.8.1 Grouping of Functions in a Layer

The functions related to a category are grouped together and made into a layer of that category. For example, process creation, deletion, saving the context of a process, and so on, may be grouped together and named as process management layer. The topmost layer provides the interface to applications of the users. The lowest layer interacts with the underlying hardware.

4.8.2 Hierarchy of Layers

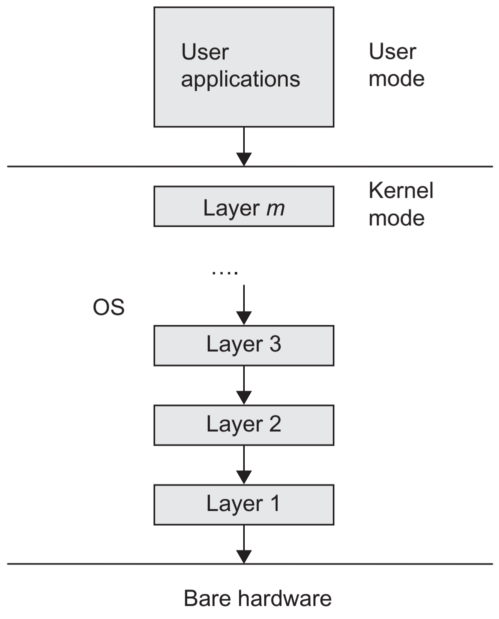


Fig. 4.8 Layered architecture

The hierarchy of layers is maintained in this architecture to reduce the complexity of interfacing among all the layers. It means that any layer cannot interface with any other layer. There is a proper hierarchy between the layers. Each layer can communicate with only layers immediately below or above it. Moreover, a layer uses the services of the layer below it. It means each lower-level layer provides the functionalities to its higher level. Thus, only adjacent layers communicate.

The layered architecture of OS came into existence due to this design. The layered architecture consists of many layers of different functionalities identified in a design. These layers have pre-defined hierarchy, and interfacing among them is simple when compared to monolithic architecture. This design has simplified the architecture of OSs. The limited interface among the layers has resulted in a simple architecture. The layered architecture provides the modularity wherein there is a defined layer for each group of functionality. Therefore, if there is a need to change a module or debug an error, it becomes easy because changes or debugging an error are localized to only one layer. It means changes made in one layer do not affect the other layers. Similarly, if we want to find some error in one layer, we are concerned with only that layer. Debugging is easy due to this localization of errors.

Another advantage in this architecture is that there is protection among different modules of different layers. Due to limited interface among layers and a proper hierarchy, no module can enter into others' area, thereby giving protection among layers and their modules. For instance, the upper layer can invoke a module only of the lower layer and does not know the addresses of data and instructions of that module. It means implementation and interfaces of a module have been separated. The upper layer needs to know how to interface with the lower module or what modules of the lower layer is to be called but need not know how those modules have been implemented. Thus, this design prevents a malicious code and corruption of data from one layer to another.

There are also some disadvantages in layered architecture. In this design when a system call appears, it needs to pass through all the layers for getting the functionality of the requested resource. Since there is hierarchy of layers and limited interaction between them, it will take some time to execute a system call due to time taken in getting the system call request from the topmost layer to the lower layer and then to the actual resource. Thus, this design may suffer from efficiency problems if there is a large number of layers. The increasing number of layers may again lead to a complex architecture. Another problem in the layered architecture is to group the modules in a layer such that the upper layer is able to invoke the modules of the lower layer. It may be difficult sometimes to isolate functionalities from one another. Then in this case, the decision of placing the modules in a fixed layer or defining the roles of each layer may be difficult.

4.9 VIRTUAL MACHINE OS

A user may have different types of requirements to execute jobs. Some jobs are batch oriented, that is, these jobs do not need attention or any interaction of the user. Some jobs require immediate attention and quick response. However, the OS structure is either batch oriented or time-sharing. It means that the same structure of the OS cannot be used for different types of requirements for executing jobs. The same happened to OS/360, which was a batch system. But users also demanded to have a time-sharing environment in the system. IBM then developed Time-Sharing System/360 (TSS/360), which was big and slow and, in turn, abandoned. It means the same structure of OS would not be suitable for providing different requirements. Later on, the system was redesigned and called Control program/Conversational monitor system (CP/CMS). Later this was renamed as Virtual Machine Facility/370 (VM/370).

The solution adopted in VM/370 to support different types of requirements of the user was to have different types of OSs. These different types of OSs would support the different functionalities desired by the user. The different OSs were realized through virtual machine concept (see Fig. 4.9). It means these OSs will run on virtual machines. If there are three virtual machines, then it means three different OSs can be supported. But these virtual machines are not extended machines as discussed in Chapter 1 but are exact copies of the bare hardware machine. Each virtual machine has the same architecture as the actual hardware. A virtual machine will have a virtual processor, memory, and I/O devices. The OSs on these virtual machines have facilities as of normal OSs on actual machines such as user/system mode, interrupt processing, and the like. However, the configuration of the virtual machine may not be the same as that of actual hardware. For instance, the size of memory will be smaller in the virtual machine. To implement the virtual machine, in fact, they are mapped on to the bare hardware machine. The services of various OSs, running on their virtual machines, are also required. This is done by the *host OS*, which is running on the bare hardware. The host OS multiplexes the virtual processors onto the actual CPU of the host computer. The host OS decides which OS to run next. This is just like how the normal OS performs scheduling of jobs on the CPU. The host OS switches

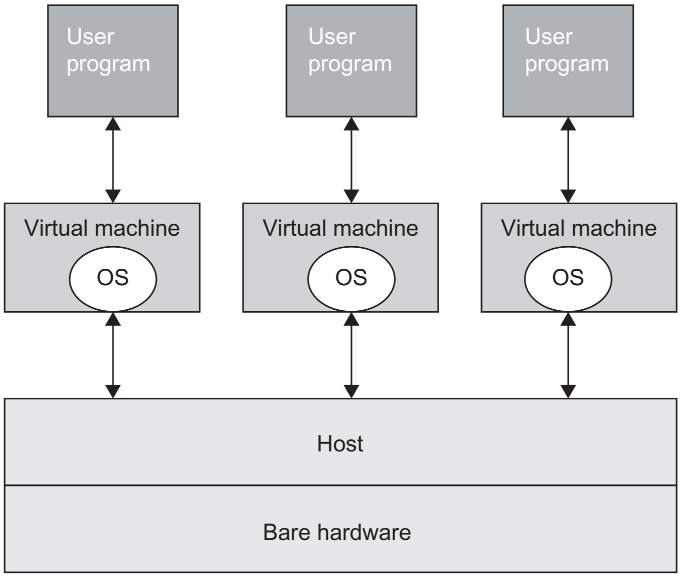


Fig. 4.9 Virtual machine architecture

the control between various OSs running on virtual machines in the same fashion as done for different jobs. In this way, the host OS maps the virtual machine onto the bare hardware and the functionality of the OS running on that virtual machine is achieved. Thus, different virtual machines may run different OSs as required. This architecture of the OS is known as *virtual machine OS*. The obvious advantage of virtual machine OS is that the same hardware is being shared to run different execution environments, that is, multi-programming, time-sharing can be on a single machine. Another advantage of these OSs is that all virtual machines are isolated and, thus, protected. Similarly, host OSs running on bare hardware are protected from virtual machines. For example, a virus in the host OS may corrupt it but cannot corrupt guest OSs.

The virtual machine concept in OSs can be seen in various systems. Some of them are discussed here. As discussed earlier, VM/370 was a virtual machine-based OS. Other versions of this system are VM/SP and z/VM. The host OS in this system (see Fig. 4.10) is a control program known as hypervisor or, generally, VM-CP. It runs on the bare hardware and creates the virtual machine environment by coordinating among various virtual machines. The VM-CP implements the virtualization of the actual hardware, including all I/O and other privileged operations. It performs the system's resource sharing, including device management, dispatching, virtual storage management, and other traditional OS tasks. Each user views the system as having a separate machine, the virtual machine. All virtual machines have their own address spaces, I/O, interrupts, and so on, just like the real machine has. Thus, virtual machines are not extended machines but limited versions of exact hardware.

The OS running on the virtual machines, sometimes called guest OSs, is known as Conversational Monitor System (CMS), which is a single-user interactive system. But any other mainstream OS can also run on virtual machine. These guest OSs are prevented from using privileged instruction by the hypervisor. However, the hypervisor simulates privileged instruction on their behalf.

Another use of virtual machines has been made in Pentium machine. On this machine, along with Windows, MS-DOS programs can be run. Intel provided a virtual 8086 mode on this Pentium architecture and in this mode, DOS programs are started up.

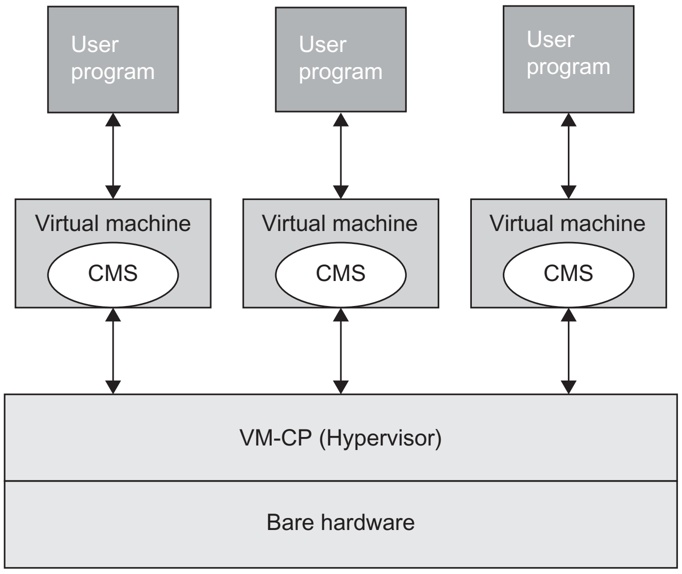


Fig. 4.10 VM/370 architecture

4.10 CLIENT–SERVER OR MICROKERNEL-BASED OS

As the computer architecture improved over time and demands from an OS increased, the size of kernel expanded beyond limit. Larger-sized kernels were more prone to errors and difficult to maintain. Whatever the architecture; the large kernel size became unmanageable and suffered from the difficulty of extensibility, efficiency, and reliability. A number of new devices have appeared and many of them have already disappeared. There is a need to add or delete their support in the OS. But due to the monolithic or layered structure of kernel, it was not easy to add or delete the modules, because there was dependency among modules or layers. In layered architecture, as the number of layers increased with more demands in the functionality of the kernel, the OS started to perform badly as compared to previous architectures. Due to this reason, Windows NT with layered architecture gave bad performance as compared to Windows95. Due to the more integrated nature of layers or modules, if one component fails, then the whole OS goes down. It decreases the reliability of the system.

To remove heavy functionalities from the kernel, it was thought that some essential functionality will remain inside the kernel known as *essential core* of the OS code. The kernel, consisting of essential core, is called *microkernel*. The components of the essential core may be process management, inter-process communication, low-level memory management, and so on. The other OS modules, which were considered as non-essential, were moved up in the user space. In this way, microkernel was designed to manage the large-size kernel by dividing it into two parts: kernel space code and user space code. The only difficulty in microkernel is to decide the essential core of the kernel. Developers have discussed features to be included inside the kernel and features to be incorporated in the user space.

The modules implemented outside the kernel in user space are called *server processes*. The application programs of the user (client programs) communicate with the server processes as shown in Fig. 4.11. The server processes provide various services like file system management, process scheduling, device management, networking, and so on. The communication between

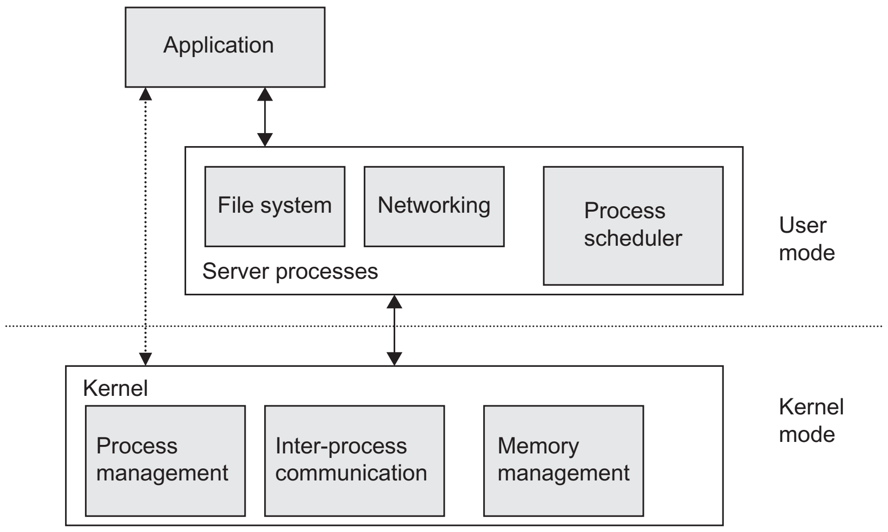


Fig. 4.11 Microkernel architecture

the client applications and server processes is through the message-passing communication method, called as Inter-process Communication (IPC). This is the reason this architecture is also known as client-server architecture of OSs. The microkernel facilitates this message-passing communication by validating the messages, passing them on to various modules and giving access to hardware. Thus, IPC is the basic mechanism in microkernel through which a service is provided. Any service is obtained by sending an IPC message to a server, and obtaining the result in another IPC message from the server. The server processes are like any other program that allows the OS to be modified simply by starting and stopping these server programs. For example, in a machine, if networking support is not required, then the server module supporting networking is not started without any further change in the OS. But if this is the case in other architectures of OSs, then the kernel needs to be recompiled after removing the networking module. Therefore, the microkernel structure is more adaptable as compared to others. Similarly, if a server module fails, it can be stopped, rather than crashing the kernel itself. In this way, microkernel is more robust and reliable.

The extensibility feature of microkernel architecture can be used for developing various types of OSs, using the same microkernel. For example, Mach microkernel has been used to develop several OSs like True64 UNIX and SPIN.

Microkernel architecture has been used in many systems. Mach was the first OS designed with this approach. Mach, developed in the 1980s, was the most successful microkernel and has been used in various commercial systems. For example, True64 UNIX and SPIN were built on Mach microkernel. The microkernel has been evolved over several generations. The first-generation microkernels (Mach, L3, and so on) were slower in nature due to IPC used for communication between kernel and servers. This inefficiency was removed in later generations of microkernel. Some examples of the latest generation, which are now faster than the first-generation microkernels, are L4, SPIN, and QNX. 'PARAS' developed by Centre for Development of Advance Computing (C-DAC), India, is another example of microkernel architecture.

4.11 EXOKERNEL

There may be the case that the performance of an application being developed is affected due to the architecture of the OS. Although, there are various options of OSs to be selected for various types of applications, it may not be possible sometimes that the required feature is available in the selected OS. For example, a file system that does not leave old data on the disk may be suitable for security-oriented application but not for a reliability-oriented application. The reason behind this is, that in the original concept of OS, the hardware has been at such high abstraction level to application developers that they, in fact, do not know about the actual hardware. They work in a convenient environment, provided by the OS, without worrying about the bare hardware details. In all the structures of the OS, it has been ensured that the developers need not worry about the configuration and limits of the hardware. This is because the OS is there for all these tasks and provides a convenient and friendly environment to the user. The OS provides a conceptual model through processes, threads, schedulers, file system, paging, virtual memory, sockets, and so on, on which a developer works. But, researchers at MIT realized that giving so much abstraction to the developer affects the performance of the application being designed on the system. It would be better for a developer if he/she decides on his/her own about what to do with resources instead of following the abstractions provided by the OS. In this way, performance of an application, not bounded by the abstractions and policies of the OS, may perform better.

To implement this idea, control of the resources need to be provided to the developers. One way is to program directly on the hardware and remove the kernel as we did in the past, without the OS. But this idea is to return to the past and we have seen how difficult life was without OSs. The compromising idea is to use a kernel with minimum functionality and providing access of resources to the developer as well. This kernel is called exokernel. The exokernel performs the job of allocation and synchronizing of resources with user jobs. But the way in which the application makes use of resources will be decided by the developer. In other words, the exokernel works as an executive for application programs such that it ensures the safe use of resources and allocates them to the applications. It means that the developer can implement the customized abstraction on the resources. The applications implemented in this way are known as *library OSs*. Library OSs (see Fig. 4.12) may request the exokernel to allocate resources like some disk blocks, memory addresses, CPU, and so on, and use these resources the way it suits the application best. In this way, the exokernel also provides the efficiency, because now there is no need to map the resources from the conceptual model provided by the conventional OS to the physical model. For example, in the MIT Exokernel Project, the Cheetah web server stores pre-formatted Internet Protocol packets on the disk, the kernel provides safe access to the disk by preventing unauthorized reading and writing, but the method in which the disk is abstracted is up to the application or the libraries the application uses.

Though the concept of exokernel has been in use since 1994, and MIT has developed two exokernels, namely Aegis and XOK, this concept has not been used in any commercial OS and research is still going on.

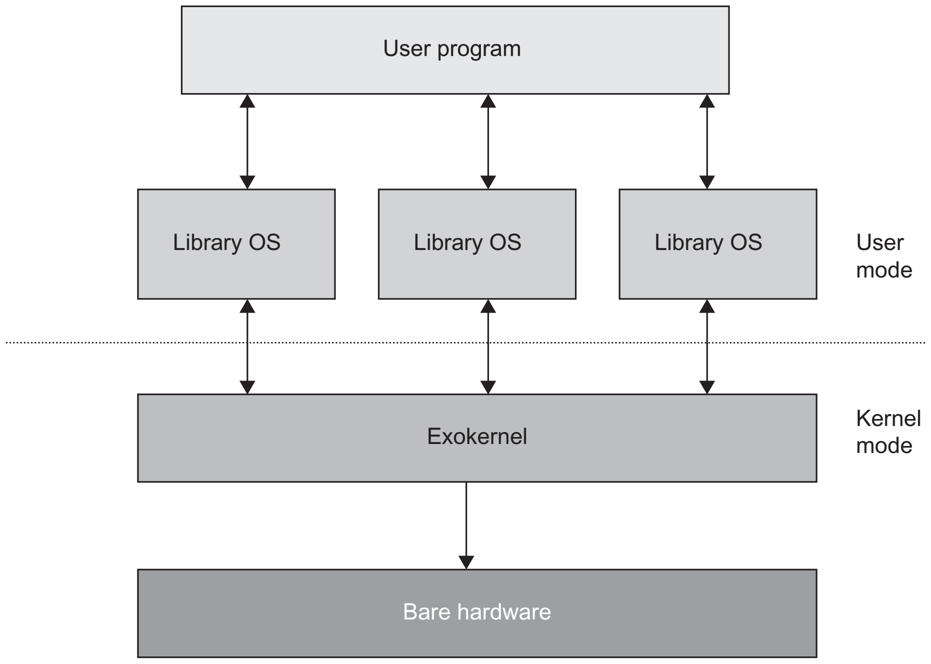


Fig. 4.12 Exokernel architecture

4.12 HYBRID KERNEL-BASED OS

In microkernel architecture, there was a cost involved in IPC for context switching from user mode to kernel mode and vice versa. Context switching will be elaborated in Chapter 5. In a component like networking, the overhead of context switching is too high. Therefore, some kernels were designed to bring back some of the components inside the kernel. Moreover, the advantages of a layered approach were also merged with microkernel architecture. This type of kernel, having the mixed approach of various structures, is known as *hybrid kernel*. Various OSs have been designed using hybrid kernels. For example, the architecture of Windows NT and Windows 2000 has been designed with the hybrid approach, taking advantage of layered as well as microkernel approaches. In order to reduce the cost of IPC in microkernel, the modules that were in the user space have been brought back inside the kernel. But these modules are still not inside the microkernel. It means the kernel's non-essential functionalities and core essential functionalities are still separate. This has been achieved by the layered concept. There are separate layers of kernel functionalities called as the executive layer and below this layer is the microkernel layer. There are basically three layers: executive layer, microkernel layer, and hardware abstraction layer. The executive layer includes high-level kernel functionalities like object management, IPC, virtual memory management, and so on. The microkernel layer provides minimal kernel functionality such as process synchronization, scheduling, and the like, as discussed in microkernel architecture. Hardware abstraction layer provides easy portability to a number of hardware architectures. In this way, a hierarchy of layers to separate the microkernel from other functionalities of kernel has been designed. But both microkernel and executive layers are in kernel mode, thereby reducing the context switch overheads.

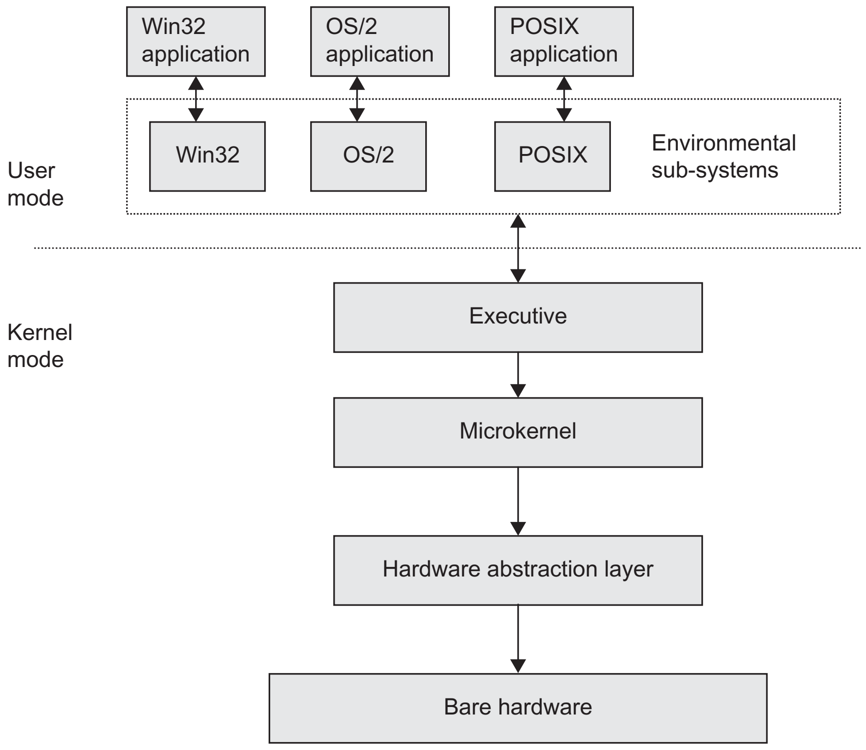


Fig. 4.13 Windows hybrid architecture

SUMMARY

The very first architecture of an OS was monolithic. With the introduction of multi-programming, it was evolved to layered architecture due to some constraints like security, debugging, and so on. The layered architecture provides the modularity wherein there is a defined layer for each group of functionality. Therefore, if there is a need to change a module or debug an error, it becomes easy to do so because changes or debugging an error are localized to one layer only. It means changes made in one layer do not affect the others. Another architecture known as virtual machine OS was designed to cater to the different needs of a user. As the computer architecture improved over time and demands from an OS increased, the size of the kernel expanded beyond limit. Larger-sized kernels were more prone to errors and difficult to maintain. Therefore, another architecture was designed, known as Microkernel-based OS. Since the OS abstracts the hardware to a user and hides the complex details of the hardware, it was realized that the performance of the application being developed may be affected somewhere, as all the decisions regarding the resources in the system are with the OS. In response to this, exokernel architecture was developed, giving some access of resources to the user. Finally, a hybrid architecture combining the merits of some OS architectures was designed.

Let us have a quick review of important concepts discussed in this chapter:

- BIOS is a software that consists of input-output functions. These functions are low-level routines that the OS uses to interface with different I/O devices like keyboard, screen, ports, and so on.
- The set of instructions needed for booting, that is, to load the OS in RAM is known as Boot software/Boot loader/Bootstrap loader.
- The instructions, which are not directly executed by the user but need to be passed to the OS, are known as privileged instructions.
- All the privileged instructions, that is, instructions that need to interact with hardware and resources, and therefore passed on to the OS for execution, are known as system calls.
- POST is a built-in diagnostic program that initializes and configures a processor and then checks the hardware to ensure that every connected device is present and is functioning properly.
- The difference between system call and a normal function call is that a system call enters the kernel but a normal function call does not. Another difference is that the system call itself does not execute anything but generates an interrupt, which changes the mode to system mode and passes the control over to the OS.
- System programs help a user in developing and executing his/her applications. System programs should not be confused with system calls. System programs are utilities programs, which help the user and may call for further system calls.
- System generation is the process of configuring the OS according to the hardware and other specifications on a particular machine.
- Monolithic systems were not suitable for multi-programming/multi-tasking environments due to the unprotected behaviour of the system. Any user job can corrupt another user's job and even the OS.
- Layered architecture provides the modularity wherein there is a defined layer for each group of functionality. Therefore, if there is a need to change a module or debug an error, it becomes easy because changes or debugging an error are localized to one layer only.
- The advantage of the virtual machine OS is that same hardware is being shared to run different execution environments, that is, multi-programming and time-sharing can be on a single machine. Another advantage of these systems is that all virtual machines are isolated and, thus, protected.
- Exokernel works as an executive for application programs such that it ensures the safe use of resources and allocates them to the applications. It means that the developer can now implement the customized abstraction on the resources.
- Hybrid architecture combines the features of microkernel and layered architectures.

MULTIPLE CHOICE QUESTIONS

1. All the privileged instructions, that is, the instructions, which need to interact with hardware and other resources, and, therefore, passed on to the OS for execution, are known as ____________.
   (a) OS procedures (c) system calls
   (b) kernel functions (d) none
2. POST is a built-in ____________ that initializes and configures a processor and then checks the hardware to ensure that every connected device is present and functioning properly.
   (a) system program (c) system call
   (b) diagnostic program (d) none
3. Boot loader contains the instructions, which, when executed, load the OS in the main memory called ____________.
   (a) bootstrapping (c) system program
   (b) POST (d) none

4. The BIOS contains a program that loads the first sector of the boot device called ____________.
   (a) boot loader (c) boot program
   (b) boot sector (d) none
5. System call is just a bridge between user programs and ____________ for executing privileged operations.
   (a) system programs (c) OS
   (b) users (d) none
6. What is the UNIX command for terminating a process abnormally?
   (a) fork (c) suspend
   (b) kill (d) none
7. What is the UNIX command for increasing the priority of a process?
   (a) priority (c) nice
   (b) lseek (d) none
8. What is the UNIX command for suspending a process?
   (a) sleep (c) pause
   (b) wait (d) none
9. What is the UNIX command for causing a process to sleep?
   (a) sleep (c) pause
   (b) wait (d) none
10. What is the UNIX command for changing the position of read/write pointer?
    (a) sleep (c) link
    (b) lseek (d) none
11. ____________ must reside in the memory forever.
    (a) resident (c) OS
    (b) transient (d) none
12. DOS is an example of ____________.
    (a) layered architecture (c) exokernel
    (b) monolithic architecture (d) none
13. The OS running on virtual machines, sometimes called guest OSs, is known as
    (a) CMS (c) VMS
    (b) TMS (d) none
14. The modules implemented outside the kernel in user space in microkernel architecture are called
    (a) servers (c) special modules
    (b) clients (d) none
15. ____________ architecture is also known as client-server architecture.
    (a) layered architecture (c) microkernel
    (b) monolithic architecture (d) none
16. TRUE64 UNIX is an example of ____________.
    (a) layered architecture (c) microkernel
    (b) monolithic architecture (d) none
17. PARAS is an example of ____________.
    (a) layered architecture (c) microkernel
    (b) exokernel architecture (d) none
18. Aegis and XOK are examples of ____________.
    (a) layered architecture (c) microkernel
    (b) exokernel architecture (d) none

REVIEW QUESTIONS

1. Define the following terms:
   (a) BIOS (b) Booting
   (c) Boot loader (d) Boot device
2. Explain all the steps of the general working of an OS.
3. What is the need of a system call? With the help of example, explain how it is executed.
4. What is the difference between a system call and a function call?
5. Explain all types of system calls using some examples.
6. What is the difference between a system call and system program?
7. What is the need of a system generation program?
8. What are the two parts in the general structure of an OS?
9. What are the shortcomings of monolithic architecture?
10. What are the advantages and disadvantages of layered architecture?
11. Explain the architecture of VM/370.
12. Explain the architecture of microkernel-based OS.
13. What is the idea behind the development of exokernel?
14. What are the good features of a hybrid-based archi tecture?

BRAIN TEASERS

1. Would microkernel architecture work well for design of an object-oriented OS? Justify your answer.
2. Design a format of message in message-passing system of microkernel architecture.
3. How is reliability increased in microkernel architecture?
4. Explore some research issues in designing an exokernel.
5. What steps would you suggest while designing an OS in order to reduce the semantic gap between user application and bare hardware?
6. Explore how UNIX has been modified to support protection.

PART II

Process Management

5 Fundamentals of Process Management

5.1 INTRODUCTION

Process is a basic term to understand the operation of an operating system. Since there are a number of user and system processes, there is a need to manage them. A running process may be interrupted at any time. Due to this concept, the processes are not in the same state forever. They change state according to an event in the system. Moreover, the state of an interrupted process needs to be saved so that it can resume its work. If a process is interrupted, another process is scheduled to be dispatched to processor for execution. Besides this, processes also need to communicate and synchronize with each other. Therefore, it is critical to manage the processes in the system from the view point of their state change, scheduling, dispatching, process switching, communication, synchronization, and so on. In this chapter, we will study these basic concepts regarding the management of processes in the system.

5.2 TERMINOLOGY

To perform a computation on the computer system, we must have a unit of work or execution for the user computation. Basically, we need a term to call all CPU activities performed by the operating system. Various terms are in use interchangeably. First of all, we take two terms: *Program* and *Job*. These two terms were in use when the batch systems developed. The term 'Program' was very common at that time and also used frequently today. A program can be considered a set of instructions in the form of modules. Thus, program is a classic term used for user's computation. Since in a batch system, there was a requirement to load and unload the magnetic tapes for various activities such as compiling, linking, loading, and so on, the term 'job' was used for performing the execution of a program through the execution of those activities. These activities were termed as a sequence of job steps as shown in Fig. 5.1. For example, the job is to execute a C program and the execution of compiler, linker, and loader programs are job steps. These job steps are in sequence, that is, loading is meaningless without the execution of linking program. Thus, *job is a sequence of single programs*. However, the terms 'job' and 'program' were used interchangeably and are also popular today as generic terms for unit of execution.

Learning Objectives

After reading this chapter, you should be able to understand:

- Difference between job, program, task, and process
- Implicit and non-implicit processes
- Process environment
- Life cycle of a process with its states and state transitions
- Implementation of processes with process control block
- Context switching
- Process switching
- Process schedulers
- Various operations on processes

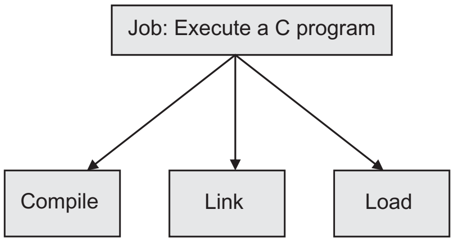


Fig. 5.1 Job as a sequence of programs

The term 'task' was used when there was a need to have concurrent execution on a single processor, that is, more than one program of a single user. For example, when a user works in Windows environment, he or she is able to open and work on multiple windows such as Word file, email, web browser, and so on. To distinguish it from the multi-programming and multiuser, the term 'task' was used and that is why it is called multi-tasking. Therefore, the term is used in the sense of multi-tasking.

The term 'process' is different from the terms 'job' or 'program.' We need to understand the nature of a program and process for this difference. A program is a set of instructions the user/programmer has written and stored somewhere. It means that a *program is a passive entity* and continues to exist at a place. On the other hand, when a program is ready for the execution, it becomes active and is known as a process. In other words, a *program in execution is called a process*. Thus, a process is an active entity with a limited span of time as compared to a program. The term 'task' is also sometimes used interchangeably with the term 'process'.

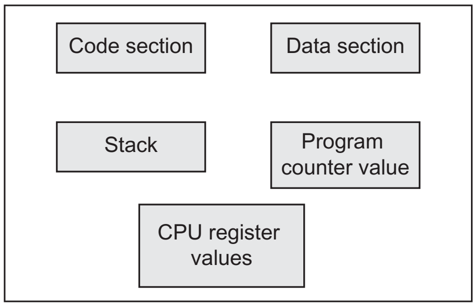


Fig. 5.2 Process environment

When a program is ready to execute or, in other words, when it becomes a process, it means that now it is able to compete for resources. Since there may be many processes ready at one time, the process needs to compete for the resources such as CPU time, memory, I/O devices, and so on. Thus, a process is eligible to compete for resources, whereas a program is not. When a process needs to execute, that is, when it gets the CPU time, it has a program counter (PC) value (initialized with process's address) also for moving to the next instruction while executing along with a code section or program code. Moreover, a data section and a stack are also allocated to a process along with other resources. When a process starts executing, the data may be stored in some CPU registers. Therefore, CPU register values are also attached to processes that are null (blank) before execution. In this way, all these together make a process's environment as shown in Fig. 5.2.

Consider an example for the difference between a program and a process (see Table 5.1). In multiuser environment, many users may open a Word program. When a user tries to open the Word file, the OS loads the Word program into memory, creating a process for the user. Now this process has separate set of resources as mentioned in Section 5.1 for execution. The process is then scheduled for execution. If another user opens the Word file, the OS again creates and schedules a separate process for the editor program. In this way, the same editor program can be invoked as separate processes. It means that there are multiple processes sharing the text editor