Operating Systems

1. Objectives and Evolution

M Suresh Babu

M Suresh Babu is currently a fourth year undergraduate student in the Department of Computer Science and Engineering, Narayana Engineering College, Nellore, Andhra Pradesh. He would like to work in operating systems, computer networks and also in Internet security concepts.

In this article we examine the objectives and functions of *operating systems,* and then we trace the evolution of operating systems from the first manual system to the current multiprogrammed and time-shared systems.

Introduction

An operating system (OS) is a program that controls the execution of an application program and acts as an interface between the user of a computer and computer hardware. The purpose of an OS is to provide an environment in which a user can execute programs in a convenient and efficient manner.

The operating system must provide certain services to programs and to the users of those programs in order to make the programming task easier. The specific services provided will, of course, differ from one OS to another, but there are some common classes of services that we identify and explore in this article. After reading this article you can understand the reasons behind development of OS and appreciate the tasks an OS does and how it does them.

What is an Operating System?

An operating system is an important part of almost every computer system. An OS is similar to a government; a resource allocator and a control program. As a government the OS performs no useful function by itself, it simply provides a good environment. As a resource allocator, it acts as manager of resources (hardware and software) and allocates them to specific programs and users as necessary for tasks. As a control program it controls the execution of user programs to prevent errors and improve use of computer.

Keywords
Operating systems, file systems, time-shared systems.

A more common definition of an operating system is the only program running at all times on the computer (called as Kernel), with all else being application programs. It is easier to define operating system by what it does than what it is.

Operating systems and computer architecture have had a great deal of influence on each other.

Objectives of Operating Systems

An operating system can be thought of as having three objectives or as performing three functions.

Convenience: An operating system makes a computer more convenient to use.

Efficiency: An operating system allows the computer system resources to be used in an efficient manner.

Ability to evolve: An operating system should be constructed in such a way as to permit the effective development, testing and introduction of new system functions without interfering with current services provided.

Evolution of Operating System

Operating system and computer architecture have had a great deal of influence on each other. To facilitate the use of the hardware, OS's were developed. As operating systems were designed and used, it became obvious that changes in the design of the hardware could simplify them.

Simple Batch Systems

When punched cards were used for user jobs, processing of a job involved physical actions by the system operator, e.g., loading a deck of cards into the card reader, pressing switches on the computer's console to initiate a job, etc. These actions wasted a lot of central processing unit (CPU) time.

To speed up processing, jobs with similar needs were *batched* together and were run as a group. Batch processing (BP) was implemented by locating a component of the BP system, called the batch monitor or supervisor, permanently in one part of

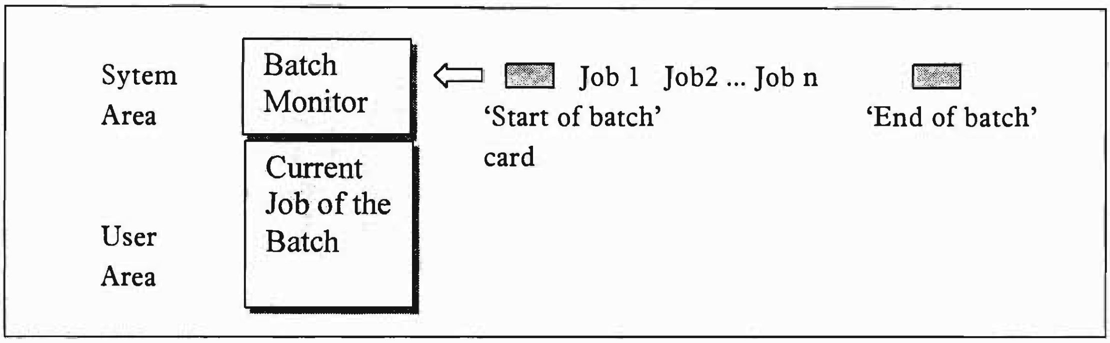


Figure 1. A schematic diagram of BP system.

computer's memory. The remaining memory was used to process a user job – the current job in the batch (*Figure* 1).

The delay between job submission and completion was considerable in batch processed system as a number of programs were put in a batch and the entire batch had to be processed before the results were printed. Further card reading and printing were slow as they used slower mechanical units compared to CPU which was electronic. The speed mismatch was of the order of 1000. To alleviate this problem programs were spooled. Spool is an acronym for *s*imultaneous *p*eripheral *o*peration *o*n-*l*ine. In essence the idea was to use a cheaper processor known as peripheral processing unit (PPU) to read programs and data from cards store them on a disk. The faster CPU read programs/data from the disk processed them and wrote the results back on the disk. The cheaper processor then read the results from the disk and printed them (*Figure* 2).

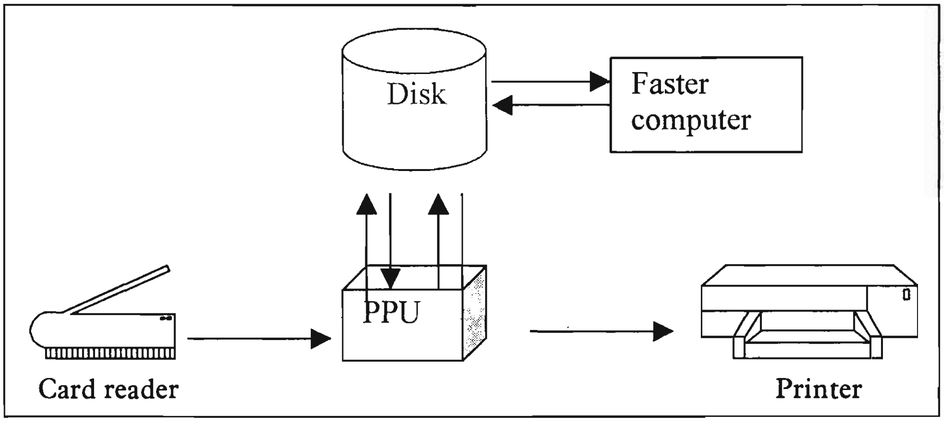


Figure 2. Spooling of programs.

Multi Programmed Batch Systems

Even though disks are faster than card reader/ printer they are still two orders of magnitude slower than CPU. It is thus useful to have several programs ready to run waiting in the main memory of CPU. When one program needs input/output (I/O) from disk it is suspended and another program whose data is already in main memory is taken up for execution. This is called multiprogramming.

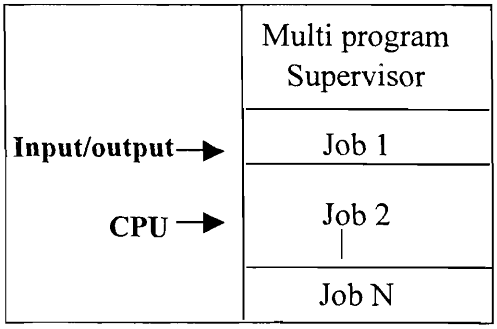


Figure 3. Multiprogramming.

Multiprogramming (MP) increases CPU utilization by organizing jobs such that the CPU always has a job to execute. Multiprogramming is the first instance where the operating system must make decisions for the user. The term MP is used to describe this arrangement (*Figure* 3).

The MP arrangement ensures concurrent operation of the CPU and the I/O subsystem. It ensures that the CPU is allocated to a program only when it is not performing an I/O operation.

Time Sharing Systems

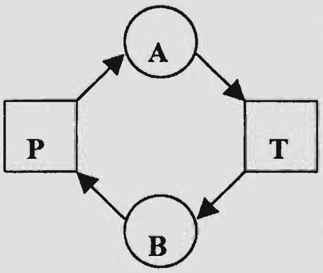
Multiprogramming features were superimposed on BP to ensure good utilization of CPU but from the point of view of a user the service was poor as the *response time*, i.e., the time elapsed between submitting a job and getting the results was unacceptably high. Development of interactive terminals changed the scenario. Computation became an *on-line* activity. A user could provide inputs to a computation from a terminal and could also examine the output of the computation on the same terminal. Hence, the response time needed to be drastically reduced. This was achieved by storing programs of several users in memory and providing each user a slice of time on CPU to process his/her program.

A time-sharing system provides a mechanism for concurrent executions, which requires sophisticated CPU scheduling schemes.

Time-sharing system provides a mechanism for concurrent executions, which requires sophisticated CPU scheduling schemes. To ensure orderly execution, the system must provide, mecha-

Box 1. Deadlock

It is possible for two or more processes (process is a program in execution) to be hung up waiting for each other. For example, process A requests permission to use the printer and is granted to it, process B then requests permission to use the tape drive, and is also granted to it. Now A asks for the tape drive, but the request is denied until B releases it. Unfortunately, instead of releasing the tape drive, B asks for the printer. At this point both processes are blocked and will remain so for ever. This situation is called deadlock.



A, B-processes
P-Printer, T-tape drive

Two fundamental approaches used for handling deadlocks are:

1. Detection and resolution of deadlocks.
2. Avoidance of deadlocks.

In the former approach, the OS detects deadlock situation as and when they arise. It then performs some actions aimed at ensuring progress for some of the deadlocked processes. These actions constitute deadlock resolution. The later approach focuses on avoiding the occurrence of deadlocks. This approach involves checking each resource request to ensure that it does not lead to a deadlock. The detection and resolution approach does not perform any such checks. The choice of the deadlock handling approach would depend on the relative costs of the approach, and its consequences for user processes [1].

nisms for job synchronization and communication, and must ensure that jobs do not get stuck in a *deadlock* (see *Box* 1).

Distributed Systems

A recent trend in computer system is to distribute computation among several processors. In the loosely *coupled systems* the processors do not share memory or a clock. Instead, each processor has its own local memory. The processors communicate with one another using communication lines, such as an ethernet.

The processors communicate with one another using communication lines and are called loosely coupled systems or distributed systems.

The processors in a distributed system may vary in size and function, and referred by a number of different names, such as sites, nodes, computers and so on depending on the context. The major reasons for building distributed systems are:

Resource sharing: If a number of different sites are connected to one another, then a user at one site may be able to use the resources available at the other.

Computation speed up: If a particular computation can be partitioned into a number of sub-computations that can run concurrently, then a distributed system may allow a user to distribute computation among the various sites to run them concurrently.

Reliability: If one site fails in a distributed system, the remaining sites can potentially continue operations.

Communication: There are many instances in which programs need to exchange data with one another. Distributed database system is an example of this.

Real-time Operating System

The advent of timesharing provided *good* response times to computer users. However, timesharing could not satisfy the requirements of some applications. Real-time (RT) operating systems were developed to meet the response requirements of such applications (for RT OS see *Box* 2).

There are two flavors of real-time systems. A *hard real-time* system guarantees that critical tasks complete at a specified time. A less restrictive type of real time system is *soft real-time* system, where a critical real-time task gets priority over other tasks, and retains that priority until it completes. The several areas in which this type is useful are *multimedia, virtual reality,* and advanced scientific projects such as *undersea exploration* and *planetary rovers.* Because of the expanded uses for soft real-time functionality, it is finding its way into most current operating systems, including major versions of *Unix* and *Windows NT* OS.

A real-time operating system is one, which helps to fulfill the worst-case response time requirements of an application. An RT OS provides the following facilities for this purpose:

1. Multitasking within an application.
2. Ability to define the priorities of tasks.
3. Priority driven or deadline oriented scheduling.
4. Programmer defined interrupts.

The areas where real-time operating systems are useful are multimedia, virtual reality, and advanced scientific projects such as undersea explorations.

Suggested Reading

[1] Silberschatz and Galvin, *Operating System Concepts*, Addison Wesley, 2000.

[2] D H Dhamdhere, *System Programming and Operating Systems*, TMH, 2000.

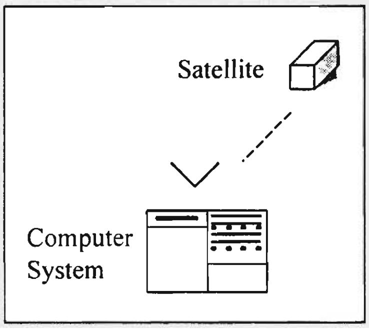
[3] Andrew S Tanenbaum, *Modern Operating Systems*, PHI, 2000.

[4] William Stallings, *Operating Systems*, PHI, 2000.

Box 2. Real-time System

A *real-time application* is an application, which requires a response within a pre-specified time from the computer system to prevent failures.

The worst-case-response-time of an application is the largest value of the response-time for which it can still function correctly. As an example, consider the real-time application depicted below. The satellite sends digitized samples at the rate of 1000 samples per second. The application program is simply required to store these samples in a file. Since a new sample arrives every millisecond, the computer must respond to every 'store the sample' request in less than 1msec. Failures to do so would result in the loss of a sample, which constitutes a failure of the application. The worst-case-response-time is thus 1ms. Other examples of real-time applications can be found in process control in chemical plants, airline reservations system, weather warning systems, etc. It can be seen that the worst-case-response-time of these systems vary greatly, in some cases it may even be a few minutes. This explodes the myth that all real-time applications require very quick response.



A *task* is a sub-computation in an application program, which can be executed concurrently with other sub-computations in the program, except at specific places in its execution called synchronization points. Multi-tasking, which permits the existence of many tasks within the application program, provides the possibility of overlapping the CPU and I/O activities of the application with one another. This helps in reducing its elapsed time. The ability to specify priorities for the tasks provides additional controls to a designer while structuring an application to meet its response-time requirements.

Acknowledgements

The author sincerely acknowledges Prof. E V Prasad, Principal and K V Raghavendra Kumar, Head of the Computer Science and Engineering Department, Narayana Engineering College, Nellore for their timely directions and helpful suggestions to increase the richness of the contents of this article.

Address for Correspondence
M Suresh Babu
C/o N Sudhakar Reddy
D. No. 16-3-1141F
Pinaki Nagar
Haranathapuram IV line
Nellore 524003
Andhra Pradesh, India.
Email:suresh_0529@
rediffmail. com