Concepts and structure of operating systems

Operating system is one of the most important types of system programs, that controls all the computer resources and provides the base upon which the application programs can be written.

A computer system consists of various hardware and software resources. The primary purpose of an operating system is to facilitate easy, efficient, fair, orderly, and secure use of these resources. It allows the users of computer system to use application software - spreadsheets, word processors, Web browsers, e-mail software, and the like. Programmers use language libraries, system calls, and program generation tools (e.g. text editors, compilers, linkers, debuggers) to develop software. Fairness is not obviously is not an issue if only one user at a time is allowed to use the computer system. However, if multiple users are allowed to use the computer system, fairness and security are two main issues to be tackled by the operating system designers.

Hardware resources include keyboard, display screen, main memory (commonly known as random access memory, or RAM), disk drive, modem, and central processing unit (CPU). Software resources include applications such as word processors, spreadsheets, games, graphing tools, picture processing tools, and Internet-related tools such as Web browsers. These applications, which reside at the topmost layer form the application user’s interface (AUI). AUI is glued to the operating system kernel via the language libraries and the system call interface. The kernel is the part of an operating system where the real work is done. The system call interface comprises a set of functions that can be used by the applications and library routines to start execution of the kernel code for a particular service, such as reading a file. The language libraries and the system call interface comprise what is commonly known as the application programmer’s interface (API).

User programs interact with the kernel through a set of standard system calls. These system calls request services to be provided by kernel. Such services would include accessing a file: open, close, read, write, execute a file; starting or
In order to use a computer system, you have to give commands to its operating system. If you use the keyboard to issue commands to the operating system, the operating system has a **character user interface (CUI)**. If the primary input device for issuing commands to the operating system is point-and-click device, such as mouse, the operating system has a **graphical user interface (GUI)**. UNIX operating system is enhanced with either X-WINDOW or GNOME graphical user interface.

Operating systems can be categorized by the number of users who can use a system at the same time and the number of processes that an operating system can run simultaneously. The criteria lead to three types of operating systems.

- **Single-user, single-process systems**: These operating systems allow only one user at a time to use the computer system, and the user can run only one process at a time. Such operating systems are commonly used for PCs. Examples of such operating systems are DOS and Window 3.1.
Single-user, multiple-process systems: As the name indicates, these operating systems allow a single user to use the computer system, but the user can run multiple processes simultaneously. These operating systems are also used on PCs. Examples of such operating systems are OS2 and Windows NT Workstation.

Multiuser, multiprocess systems: These operating systems allow multiple users to use the computer system simultaneously, and every user can run multiple processes at the same time. These operating systems are commonly used on computers that support multiple users in organizations such as universities and large business. Examples of such operating systems are LINUX and Windows NT Server.

Processes

Introduction to Processes. A key concept in all operating systems is the process. A process is basically a program in execution. It consists of the executable program, the program’s data and stack, program counter, stack pointer, and other registers and all the other information needed to run the program.

All modern computers can do several things at the same time. While running a user program, a computer can also be reading from a disk and printing on a terminal or printer. In a multiprogramming system, the CPU also switches from program to a program, running each for tens or hundreds of milliseconds.

In LINUX, the time a process in the CPU burst before it is switched out of the CPU is called quantum. The quantum is very short: 100msec for a typical LINUX system.

When the CPU is free, the kernel uses an algorithm to decide which process gets to use the CPU time. The technique used to choose the process that gets to use the CPU is called CPU scheduling. In LINUX the priority number is assigned to each process, and the process that has the highest priority gets to use the CPU next. Usually, the process that enters the system first is assigned the highest priority; the result is called a first-come, first-serve scheduling algorithm. Another scheme is to assign the highest priority value to a process, or a process that spends most of its time performing I/O operations. Processes, which spend most of their time performing I/O operations are known as I/O bound processes. An example of an I/O bound process is a text editor such as vi.

At any instant of time, the CPU is running only one program, in the course of 1 second, it may work on several programs, thus giving the users the illusion of parallelism. In this model, all the runnable software on the computer often including the operating system, is organized into a number of sequential processes, or just processes for short. So, a process is just an executing program, including the current values of the program counter, registers, and variables.
Conceptually, each process has own virtual CPU. In reality, of course, the real CPU switches back and forth from process to process, but to understand the system, it is much easier to think about a collection of processes running in parallel. The rapid switching back and forth is called **multiprogramming**.

In UNIX, processes are created by the `fork` call, which creates an identical copy of the called process. After the fork call, the **parent** continues running, in parallel with the **child**. The parent then may fork off more children, so at any instant it may have several executing children. The children can also execute fork, so it is possible to get a whole tree of processes, arbitrary deep.

**Process States.** Sometimes one process may generate some output that another process uses as input. Depending on relative speeds of the two processes, it may happen that one is ready to run, but there is no input waiting for it. It must then block until some input is available. It is also possible for a process that is conceptually ready and able to run to be stopped because the operating system has decided to allocate the CPU to another process for a while. These two conditions are completely different. A state diagram showing the three states a process are shown below:

1. Running (actually using the CPU at that instant)
2. Ready (Runnable; temporarily stopped to let another process run)
3. Blocked (unable to run until some external event happens)

In the first two cases the process is ready to run, only in the second one, there is temporarily no CPU available for it. The process in the third state cannot run, even if the CPU has nothing else to do.

![State Diagram](image)

Transition 1. Process blocks for input.
Transition 2. Scheduler picks another process.
Transition 3. Scheduler picks this process.
Transition 4. Input becomes available.

Four transitions are possible among these states. Transition 1 occurs when a process discovers that it cannot continue. More commonly, when a process reads from special
In LINUX, a process can be in one of the five states: ready, running, waiting, swapped, and zombie. The ready and running states are same to those in generous operating system. We say a process is in waiting state if it is waiting for some external event, such as I/O, etc. A process is swapped if it is ready to run, but it has been temporarily put on the disk; perhaps it needs more memory and there is not enough available at this time. When the parent process terminates before it executes the exit call, it becomes zombie. The process finishes and finds that the parent is not waiting. The zombie processes are finished for all practical purposes and do not reside in memory. However, they may occupy some of the kernel resources and cannot be taken out the system.

Every LINUX process has several attributes, including owner’s ID, process name, process ID (PID), process state, PID of parent process, and the length of the time the process has been running. The ps command can be used to view the attributes of processes running on a system.

If you want to monitor the CPU activity in real time, you can use top command. It displays the status of the most CPU-intensive tasks on the system and allows you to manipulate processes interactively.

Owner of the process or supervisor may delete process using kill command.

LINUX is for several activities related to process and job management, including process creation, process termination, running processes in the foreground and background, suspending processes, and switching processes from foreground to background and vice versa.

When you type a command and hit <Enter>, the shell executes the command and returns by displaying the shell prompt. While your command executes, you do not have access to your shell and therefore cannot execute any commands until the current command finishes and the shell returns. When command executes in this manner, we say that they execute in the foreground. More technically, when a command executes in foreground, it keeps control of the keyboard and the display screen.

At times you will need to run a LINUX command that takes long time to finish and you will want to do other job. This capability is called running the command in the background. You can run a command in the background by ending the command with an ampersand (&).

Last updated on September 26 2005