Operating System Overview: Part 1

This part aims to provide an overview of operating system principles and its history.

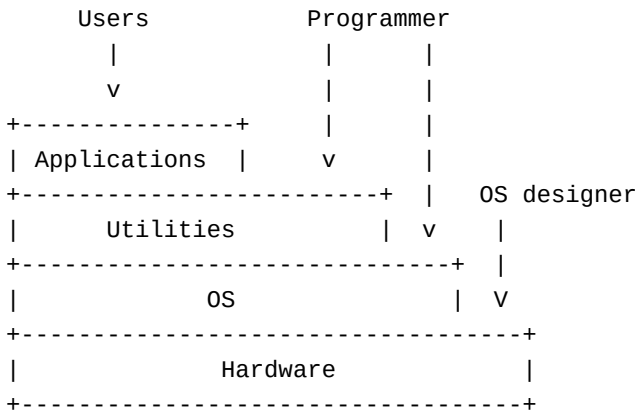
We talk about the objectives of operating system first, then look at how operation systems evolve to get closer to those goals step by step, and finally summarize the major progresses that have been made till now.

1 Objectives and functions

1.1 OS as a user/computer interface - Usability

The reason for an operation system to exist is to make computers more convenient to use. An OS aims to wrap the underneath hardware resources and provides services to end users in a systematic way. These services may be divided into two types: services directly available for end users through all kinds of I/O devices, such as mouse, keyboard, monitor, printer, and so on; and services for application programs, which in turn provides services for end users.

If we look on these services as interfaces among different components of a computer system, then the following hierarchical architecture may be obtained:



It is also common to consider `Utilities` and `Applications` that are distributed together with an OS parts of the OS, but obviously they are not essential. Utilities are usually called libraries or APIs, providing frequently used functions for the upper-level applications.

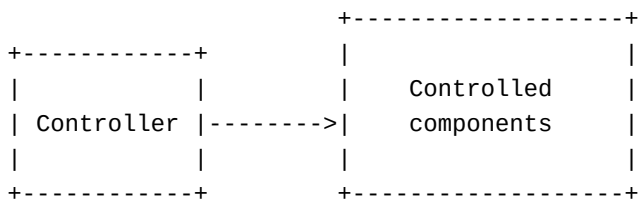
From the point of view of end users, a computer system consists of a variety of applications they may use. An application is developed by programmers in a programming language. The higher level the utilities are, the easier and more comfortable it is for programmers to code in the corresponding programming language; or the lower, the harder. In an extreme case, the assembly language is almost the same as machine instructions except that mnemonic symbols are used to replace binary strings or opcodes. In this kind of language, programmers have to deal with an overwhelmingly complexity of manipulating computer hardware. On the contrary, in a higher-level language, more user-friendly APIs are available, e.g. opening a file by calling

```
open("C:/filename.txt", "rw")
```

1.2 OS as resource manager - Efficiency

It is not the OS itself but the hardware that makes all kinds of services possible and available to application programs. An OS merely exploits the hardware to provide easily accessible interfaces. Exploitation means management upon the hardware resources, and thus also imposes control upon or manages the entities that use the services so that the resources are used efficiently. In the classes later on, we will discuss this aspect, including process scheduling, memory management, I/O device management, etc.

One thing worth mentioning here is that, different from other control systems where the controlling facility, the controller, is distinct and external to the controlled parts, the OS has to depend on the hardware resources it manages to work.



As we know, an OS is in nature a program, consisting instructions, thus it also needs CPU to execute instructions so as to function as a controller, and main memory to hold instructions for CPU to fetch. At the same time, the OS has to be able to relinquish and regain later the control of CPU so that other programs can get chance to run but still under the control of the OS (An analogy to this is that an administrator of an apartment building might live in the same building himself). By utilizing the facilities provided by hardware, the OS may schedule different processes to run at different moments and exchange the instructions and data of programs between external storage devices, like hard disks, and main memory. These topics will be covered as the course proceeds.

1.3 Evolution of OS - Maintainability

It does not suffice to simply consider an operating system an unvariable unit. An OS may evolve while time elapses due to the following reasons:

- **hardware upgrades or new types of hardware**: With hardware technologies development, the OS also needs to upgrade so as to utilize the new mechanisms introduced by new hardware. For example, Pentium IV extended instruction set of Pentium III for multimedia applications and internet transmission. An OS designed for the previous versions of Intel x86 series will have to be upgraded to be able to accommodate these new instructions.
- **new services**: An OS may also expand to include more services in response to user demand.
- **fixes**: No software is perfect, and any program may contain more or less bugs or defects, thus fixes should be made from time to time. Microsoft Windows is a vivid example of this kind.

These situations all require OS designers to build an OS in the way that the system can be maintained and upgraded easily. All the common software design techniques may be applied to an OS, such as modularization. With modularization, the OS is split into multiple modules with clearly defined interfaces between them. Thus, as long as the interfaces are left untouched, each single module may be upgraded independently.

2 The evolution of operating systems

To better understand the requirements for an operating system, it is useful to consider how operating systems have evolved over the years.

2.1 Serial processing

The earliest computer system has no OS at all, and is characterized as *serial processing* because users have to reserve time slots in advance, and during the alloted period, they occupy the computer exclusively. Thus the computer will be used in sequence by different users.

These early systems presented two major problems:

1. Users may finish their tasks earlier than you have expected, and unfortunately the rest time is simply wasted. Or they may run into problems, cannot finish in the allotted time, and thus are forced to stop, which causes much inconvenience and delays the development.

2. In such systems, programs are presented by cards. Each card has several locations on it, where there may be a hole or not, respectively indicating 0 or 1. Programs are loaded into memory via a card reader. With no OS available, to compile their programs, users have to manually load the compiler program first with the user program as input. This involves mounting, or dismounting tapes or setting up card decks. If an error occurred, the user has to repeat the whole process from the very beginning. Thus much time is wasted.

2.2 Simple batch systems

To improve the utilization of computer systems, the concept of a batch operating system was developed later on. The central idea is the use of a piece of software known as the *monitor*. With it, users don't have direct access to the computer systems any longer; instead, the operator of the system collects the user programs and batches them together sequentially for use by the monitor.

To process user programs, the monitor first has to be loaded into memory. Then it reads in programs one at a time from the input devices. As each program is read in, it will be placed in the user program area of main memory, and control is passed to this program. When the execution of the program is completed, it returns control to the monitor, which moves on to process the next program.

To assist in this, a special language is used, called *job control language* or JCL (At that time, each user program was called a job), which provides instructions to the monitor. For example, the following is a FORTRAN program with JCL commands.

```
$JOB                                $RUN
$FTN                                ...
...                                 Data
FORTRAN instructions                ...
...                                 $END
```

`$JOB` indicates the beginning of a job. `$FTN` tells the monitor to load the FORTRAN compiler, which generates object code from the FORTRAN source program. The monitor regains control after the compile operation. `$RUN` makes control transferred from the monitor to the current program, which works on the data following until a successful or unsuccessful completion. The monitor then may move on to another job. To much extent, JCL instructions are similar to

the command sequence in a DOS batch file or a UNIX shell file. The latters tell the operating systems to run multiple commands automatically.

The advantage of this mode is that the monitor automates the execution of multiple jobs thus much time is saved by avoiding manual operations.

2.3 Multiprogrammed batch systems

Even with the automatic job processing by a monitor, the processor is still often idle. The problem is actually what we have discussed before regarding programmed I/O. That is a program may have to wait for I/O operation to finish and thus leads to the processor's idling. The solution is to run multiple programs concurrently during a certain period so that whenever the current program has to wait for I/O devices, control may be transferred to another program. If needed, a third program may be loaded, or even more. This scheme is called *multiprogramming* or *multitasking*.

With multiprogramming, the utilization of processor is greatly improved, but it has its own problems. To run multiple programs concurrently, the memory should be organized properly so that each program has its own space and does not invade others'. What's more, at some moment, there may be more than programs ready to run. Thus some form of scheduling is needed to obtain better performance.

2.4 Time-sharing system

With multiprogramming, the overall system is quite efficient. However a problem remains. That is those jobs that come late in the batch job list won't get chance to run until the jobs before them have completed, thus their users have to wait a long time to obtain the results. Some programs may even need interaction with users, which requires the processor to switch to these programs frequently.

To reach this new goal, a similar technique to multiprogramming can be used, called *time sharing*. In such a system, multiple users simultaneously access the system through terminals, with the operating system interleaving the execution of each user program in a short burst of computation. For example, suppose a computer system may have at most 10 users at the same time, and the human reaction time is 200 ms. Then we may assign $200/10 = 20ms$ CPU time to the user programs one by one in a cyclic manner, thus each user will be responded within the human reaction time so that the computer system seems to service the user program itself.

The following table gives the difference between the batch multiprogramming and time sharing:

	Batch multiprogramming	**Time sharing**
Principal objective	Maximize processor use	Minimize response time
Source of directives to operating system	Job control language commands provided with the job	Commands entered at the terminal

Table 1: Batch multiprogramming versus time sharing