Introduction (Chapter 1)

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

Mainframe OSs are designed primarily to optimize utilization of hardware,

PCs OSs support games and business applications, and everything in between,

Handheld computer OSs provide an easy (convenient) interface with the computer to execute programs.

So some OSs are designed to be convenient, other to be efficient, and others to be a combination of the two.

A computer has four components: hardware, OS, application programs, and the users.

The **hardware** (consisting of the CPU, memory, and I/O devices) provides the basic computing resources,

the **application programs** (e.g., word processors, spreadsheets, web browsers, compilers, etc.) define the ways in which these resources are employed by the users.

The OS controls and coordinates the use of the hardware among the various application programs for the various users.

The components of a computer system are its hardware, software, and data. The OS provides the means for the proper use of these resources in the operation of the computer system.

OS can be explored from two viewpoints: the **user** and the **system**.

User Viewpoint

Most users sit in front of a PC (consisting of monitor, keyboard, mouse, and system unit). Such a system is designed for one user to monopolize its resources, to maximize the work or play that the single user is performing. In this case, the OS paradigm is **ease of use**, with some attention paid to performance, and none to resource utilization.

Some users sit at a **mainframe or minicomputer**, with other users accessing the same computer through other terminals, sharing resources and information. The OS is designed to maximize **resource utilization**, to assure that CPU time, memory, and I/O devices are used efficiently and fairly.

Other users sit at **workstations**, connected to the network and other workstations and servers. These users have dedicated resources at their disposal, but they also share printers, compute time of servers, network, etc. Therefore, the OS is designed to compromise between individual usability and resource utilization.

System Viewpoint

The OS is intimate with the hardware. We can view the OS as a **resource allocator**; the OS acts as a manager of the CPU, memory, I/O devices, etc.

The OS must satisfy (sometimes conflicting) requests for resources, made by users and programs.

The OS can also be viewed as a **control program** that manages the execution of user programs to prevent errors and improper use of the computer. It is especially concerned with the operation and control of I/O devices.

We have no completely adequate definition of OSs; they exist because they are needed to create a usable computing system. However, a common definition is that the OS is the one program running at all times in the computer (usually called the **kernel**), with all else being application programs.

The matter of *what is an OS* is important: in 1998 the US Dept of Justice, filed suit against Microsoft, in essence claiming that Microsoft included too much functionality in its OS and prevented competition from other vendors.

Convenience and **efficiency** are (sometimes contradictory) goals for an OS.

OSs have evolved over time. For example, UNIX started with a keyboard and a printer as its interface, and over time **graphic user interfaces (GUI)** were added, allowing UNIX to be more convenient while still concentrating on efficiency.

OSs and computer architecture have influenced each other a great deal: to facilitate the use of hardware, researchers developed OSs. Users of OSs have then proposed changes to hardware to simplify them.

Mainframe systems

Mainframe computer systems were the first computers used to tackle many commercial and scientific applications.

These early computers were physically enormous machines run from a **console***; the common input devices were card readers and tape drives.

The common output devices were line printers, tape drives, and card punchers. The user did not interact directly with the computer system.

*The Oxford Dictionary defines a console as: 1 panel for switches, controls, etc. 2 cabinet for a television etc. 3 cabinet with a keyboard and stops of an organ.

The user did not interact directly with the computer; the user prepared a job (consisting of a program, the data and some control information about the nature of the job), and submitted it to the computer operator. The job was usually in the form of punch cards. Sometime later (minutes, hours, or days), the output appeared. The output consisted of the results of the program, as well as a dump of the final memory and register contents for debugging.

The OS in these early computers was simple: its major task was to transfer control automatically from one job to the next. The OS was always resident in memory.

To speed operation, operators **batched** together jobs with similar needs, and run them through the computer as a group.

In the batch execution environment, the CPU is often idle, because the speeds of the mechanical I/O devices are intrinsically slower than are those of electronic devices.

Even a slow CPU works in the microsecond range, while a fast card reader might read 1200 cards per minute. Thus, the difference in speed between the CPU and its I/O devices may be three orders of magnitude or more.

Over time fast disks were introduces, but CPUs have also become faster, so the problem, instead of being solved, has exacerbated.

The introduction of disk technology allowed the OS to keep all jobs on a disk, rather than on a serial card reader. With direct access to jobs, the OS can perform **job scheduling**, to use resources and perform tasks efficiently.

The most important aspect of job scheduling is **multiprogramming**. A single user cannot keep the CPU and I/O devices busy at all times; multiprogramming increases CPU utilization by organizing jobs so that the CPU has always one to execute.

IDEA: the OS keeps a set of jobs in the memory–a subset of the jobs in the **job pool** (the entire job pool may not fit in the memory). The OS executes a job until it has to wait for some I/O operation to complete. Instead of keeping the CPU idle, the OS switches to another job. Eventually, the first job finishes waiting and gets the CPU back.

Multiprogramming OSs are fairly sophisticated; making the decisions of which job gets to be executed when is called job scheduling.

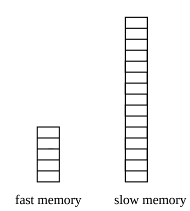
Time sharing (or **multitasking**) is a logical extension of multiprogramming. The idea is that the CPU executes multiple jobs by switching among them, but the switches occur so frequently that the user(s) can interact with each program while it is running.

An **interactive** computer system provides direct communication between the user and the system. The user gives instructions to the OS (with keyboard or mouse) and waits for immediate results. The **response time** should be short (1 second or so).

A time sharing OS allows many user to use the same computer; since the switching between jobs is fast, each user has the impression that the computer is completely dedicated to them.

Paging

Consider the following two-level **virtual memory system**:



Each level (slow and fast) can store a number of fixed-size memory units called **pages**. The slow memory stores N pages, and the fast memory stores $k < N$ pages.

Given a request for page p_i, the system must make the page p_i available in the fast memory.

If p_i is already in the fast memory (called a **hit**), the system need not do anything. Otherwise, on a **miss**, the system incurs one **page fault**, and must copy the page p_i from the slow memory to the fast memory.

In doing so, the system is faced with the problem of which page to evict from the fast memory to make space for p_i.

In order to *minimize* the number of page faults, the choice of which page to evict must be made wisely.

Examples of pairs of fast and slow memory are:

RAM and Disk

Cache and RAM

Because of its important role in the performance of almost every computer system, the paging problem has been extensively studied since the 1960s.

Often we will refer to the fast memory as the cache.

A large variety of page replacement algorithms can be found in existing computer systems:

LRU	(Least-recently-used)
CLOCK	(Clock-replacement)
FIFO	(First-in/First-out)
LIFO	(Last-in/First-out)
LFU	(Least-frequently-used)
LFD	(Longest-forward-distance)

All these algorithms, except for the last one, are **on-line algorithms**; that is, they are algorithms that make decisions based on past events, rather than the future.

The last algorithm, LFD, replaces the page whose next request is the latest. Thus, LFD requires full knowledge of future requests, and hence it is called an **off-line algorithm**.

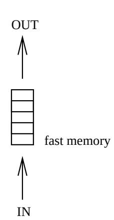
The traditional approach to studying online algorithms falls within the framework of **distributional** (or **average-case**) complexity, whereby one hypothesizes a distribution on event sequences and studies the expected payoff per event.

We are interested in the (more recent) approach of **competitive analysis**, whereby the payoff of an on-line algorithm is measured by comparing its performance to that of an *optimal offline algorithm*. Competitive analysis thus falls within the framework of **worst-case** complexity.

FIFO Page Replacement
(Section 10.4.2)

When a page must be replaced, the *oldest* page is chosen.

It is not necessary to record the time when a page was brought in; all we need to do is create a FIFO queue to hold all pages in memory:



The FIFO algorithm is easy to understand and program, but its performance is not always very good. It suffers from the so called **Belady's Anomaly**.

Suppose that we have the following sequence of page requests: $1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5$. Then, we have more page faults when $k = 4$ than when $k = 3$. That is, we have more page faults with a bigger fast-memory when using FIFO!

$1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5$