# CPS221 Lecture: Operating System Functions and History

last revised 7/28/14

*Objectives*

1. To overview the functions of an operating system
2. To survey the history of operating systems
3. To survey the modern “operating system landscape”
4. To introduce key notions: multiprogramming, preemption

*Materials:*

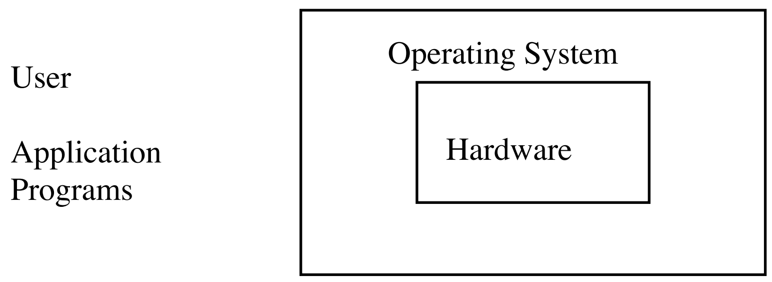
1. Projectable pictures of punched card and punched card reader
2. Projectable of stacked job batch monitor memory and input tape

## I. Introduction

A. Operating systems play a central role in the operation of most computer systems. In fact, when one speaks of a computer system, one often refers to the operating system - e.g. a “Windows machine” or a “Linux machine” or a “Macintosh” (Only in the latter case does the name say anything about the hardware, and only because MacOS only runs on machines made by Apple).

B. The operating system typically performs several kinds of functions.

1. The operating system directly controls the hardware. The user and the application programs do not see the hardware directly, but only through the operating system. In essence, what the operating system does is to create a layer of abstraction around the basic hardware, such that the user does not see the actual underlying machine, but rather a “virtual machine” that the operating system provides:



2. Another responsibility is to manage a collection of processes.

   a) A process is a program in execution.

      Note the difference between a program and a process - a process is a program in execution.

      (1) When a program is not running, there is no process corresponding to it.

      (2) When a program is running, there is generally a one-to-one correspondence between the program and the process running it.

         (a) However, it is possible that some programs, when run, may give rise to multiple processes.

         (b) Conversely, a single process may run different programs over the course of its life.

   b) Actually, on most systems it is possible for a process to be decomposed into a set of threads, each of which has its own set of register values and state, though all share the same memory. We will discuss this later.

   c) At any given time, a system may running several types of processes.

(1) Some processes are providing service directly to users - e.g command interpreters, application programs, utilities.

Example: as I am writing this, at least three processes I am aware of are currently running on my computer - one running my word processor, my email program, and a web browser.

(2) Some processes may be server processes providing services to other computers on the network.

(3) Others processes are providing operating system functions that the user may not be consciously aware of.

d) The operating system generally provides a number of services concerned with the management of processes - e.g. services that create processes, allow processes to terminate (either normally or in error) and (sometimes) that allow processes to communicate with or control other processes.

3. Another major task of the operating system is resource allocation. There is a sense in which the various processes on the system are in competition for various system resources, and it is the operating system's task to apportion them appropriately.

a) One such resource is the CPU.

(1) On a one-CPU system, only one process can actually be executing on the CPU at a time. (If the system has multiple CPU’s or is multicore, the number may be greater than one but still typically much less than the number of processes in existence).

(2) If there are multiple processes that could use the CPU, the operating system must determine which process gets it, and for how long.

b) Memory is another resource to be allocated. Each process needs one or more regions of memory to hold its code and data.

c) Another resource to be managed is the various IO devices.

(1) One user at a time devices (e.g. the keyboard on a PC)

(2) The file system

(3) Network connections.

4. Another operating system task is providing support services needed by processes. These include:

a) Support for IO operations, so that application programs can be shielded from hardware-specific details. This can be very sophisticated on windowing systems, where a single physical display is shared by multiple processes through individual windows that can overlap one in various ways.

b) File system management (directory structures, etc.)

c) Networking to other systems.

5. Finally, the operating system should also guarantee processes protection from interference by other processes.

C. Today's operating systems are among the largest, most complex, most versatile pieces of software on the market. To a large extent, the operating system carries the burden of ensuring efficient, reliable, secure availability of the computer system to its users. However, this was not always the case. Operating systems have undergone a process of evolution from the earliest days of computing - when there were essentially no operating systems - through fairly small and simple one-user-at-a-time systems, through increasingly complex multi-user systems, to their present role.

## II. **Operating System History**

A. We want to spend some time surveying some of the highlights of this process of development - for several reasons:

1. One way to get a handle on the complexity of modern operating systems is to see how they have developed one layer at a time. If we picture an operating system as an onion, then our historical survey will help us to examine the construction of the onion one layer at a time.

2. The proliferation of small, high powered computers has brought us full-circle in many areas of Computer Science, including operating systems.

   For example, a modern cell phone has far more computational power than "mainframe" computers of a few decades ago. A cell phone operating system capability (multitasking) at one point become an issue in competitive battles between Apple's IPhone and various cell phones running Google's Android.

3. Vestiges of this history still occur in sophisticated systems.

B. As an overview, we can note several key stages in the evolution of modern operating systems, originally driven by a desire to make the most efficient use of an expensive resource. (Early computers cost in the 100's of 1000's of $ or more, though that's no longer an issue!)

1. One user systems on a "bare" machine

2. Stacked job batch processing systems.

3. Multiprogrammed batch processing systems.

4. Timeshared systems.

5. The development of personal computers

6. The development of handheld devices such as smart phones and tablets.

7. Operating systems for embedded and other real-time systems

C. We will also want to take a quick look at the modern "operating system landscape", which reflects this history, as we shall see.

## III. One User Systems on a Bare Machine (No Operating System)

A. In the earliest days of computing, there were no operating systems per se. This meant that the user ran on a "bare machine".

B. It might be possible for an individual user to use such a machine, but for efficiency's sake it was more common to have a trained operator who would run user programs one at a time.

C. Systems without operating systems are still used today for simple embedded systems - such as the various systems embedded in an automobile or appliance.

## IV. Early stacked job batch systems

A. The forerunner of the modern operating system became necessary with the development of disk storage.

1. Disk drives (with typical capacities of a few megabytes!) would be used for storing commonly used programs such as compilers and libraries.

2. User programs would be submitted as decks of punched cards, which could be stacked in a physical card reader

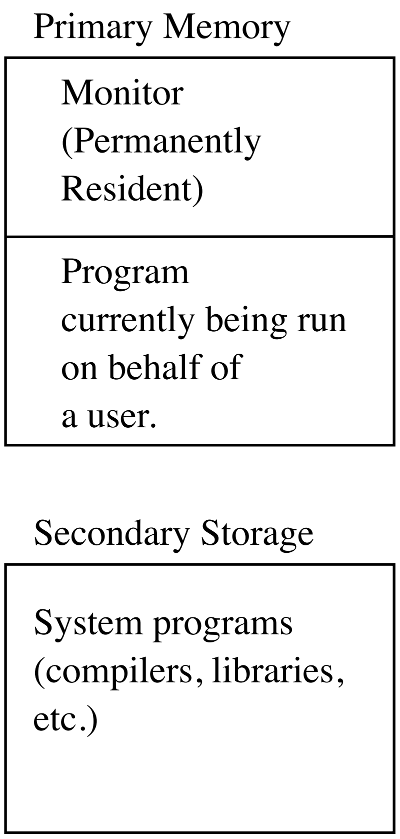
   PROJECT: Punched card, Card reader picture

   (Alternately the cards could be read offline onto a magnetic tape for faster reading by the computer)

B. In such a system, it became necessary to put more of the burden for system control on the machine, since multiple jobs were stacked behind each other in the card reader or on tape.

1. Each user job would include a series of control cards, identifying the user and the task he wished performed:

2. The program that managed the system was known as the monitor. It would be loaded when the system was powered up, and would remain resident in a designated region of memory at all times. This kind of monitor is commonly called a stacked job monitor - because jobs are processed as if they were physically stacked one behind the other in a card reader (even if, in fact, the stacking was on tape.)


Primary Memory
Monitor (Permanently Resident)
Program currently being run on behalf of a user.
Secondary Storage
System programs (compilers, libraries, etc.)

Layout of input tape

```
$JOB user id
$FORTRAN
FORTRAN source program
$LOAD
$RUN
data for program
$EOJ
$JOB user id
$LOAD application on
disk
$RUN
data for program
$EOJ
```

PROJECT. Walk through diagram; then project steps in executing jobs

3. The earliest stacked job monitors were developed by users in the mid 1950's. By the late 1950's, it was the norm for manufacturers to furnish an operating system with large machines.

4. Eventually, the command language recognized by the monitor grew into a very sophisticated language known as JCL (job-control language) that is still used on mainframe computers. This is, in effect, the **user interface** of the operating system.

5. Stacked job systems necessitated the development of the first **protection** mechanisms to protect against such problems as a program reading the program just after it as data, or going into an infinite loop and stopping other programs from being run.

6. Although stacked job systems are a thing of the distant past, relics of them remain in the “batch processing” facilities that exist in modern systems (e.g. Windows .bat files or Unix shell scripts).

V. **Multiprogramming**

A. The stacked job batch system we have described achieved an improvement in system efficiency by eliminating the time required for the human operator to load each job on the system. In its simplest form, though it left a lot to be desired in terms of overall utilization of the system resources, because when a program was using one resource, the other resources would be idle.

A case of particular concern - given high hardware costs - is that the CPU would sit idle while a job was performing a relatively slow IO operation.

B. Multiprogramming emerged in the early 1960's

1. In a multiprogrammed system, several processes are on the system at a given time, with the hope that one might be ready to use the CPU at the same time others are performing IO operations.

2. This has complexity ramifications, of course.

   a) It became necessary to provide additional memory to accommodate the additional processes.

   b) Further, the protection problem became more complex, since it is now necessary not only to protect the monitor from accidental (or deliberate) alteration by a user job, but also to protect user jobs from one another.

C. A key concept in multiprogrammed systems is the notions of **priority and preemption** to handle the commonly-occurring case where multiple jobs are able to use the CPU at the same time.

1. Different jobs could be given different priorities based on their behavior or their importance.

2. Suppose a higher priority job could not use the CPU because it was awaiting the completion of an IO operation, so a lower priority job was allowed to use the CPU. When the IO operation completed, the higher priority job might be allowed to preempt the lower priority one.

## VI. **Timesharing**

A. One problem with the systems we have considered thus far is that it was not possible for a user to interact with a program while it is running.

B. However, interaction was necessary for certain kinds of applications, such as airline reservations. One of the most important early large-scale software systems was the American Airlines SABRE system, which connected agents all over the country to a common data base of reservations. Inquiries keyed in on a terminal would receive a quick response. The underlying software, however, was not multiprogrammed, but rather a single package of programs that divided its attention among many terminals.

1. Such a system is called a transaction processing system, and is still common today in such areas as airline reservations, bank ATM's etc.

2. A similar idea exists with various server programs (web, email, etc ) in which a single program may respond to requests coming from many different users.

C. Shortly after multiprogramming appeared, efforts were begun to make interactive general-purpose computing possible through the use of timesharing.

1. A timeshared system allows interactivity by having the jobs being multiprogrammed do IO not from a (virtual) card reader and to a (virtual) printer but rather from and to a terminal with a live user in front of it.

2. The monitor used in a stacked job system is effectively replaced by a command interpreter that now served as the **user interface** by allowing the user to enter commands interactively.

3. A timer is now used to divide the available CPU time between different users, with each process that is ready to use the CPU being given a slice of the CPU’s time. If the CPU is fast enough, this can give all users the illusion of having a CPU that is dedicated to them.

   So to the notion of priority preemption we add the notion of **timer preemption**.

4. Timeshared systems are more difficult to implement than multiprogrammed batch systems or transaction processing systems, for several reasons:

   a) The typical interactive process has much more IO wait time than a batch process. With a batch process, all of the input data is available when the process starts; while an interactive process has to wait for the user to type each line - often with some thinking time in between lines. Therefore, to keep the CPU busy it is typically necessary to multiprogram more jobs so that hopefully at least one is able to use the CPU.

b) In a batch system, it is permissible for a particular job to go for a long time without service from the system, so long as it completes within a reasonable time. In an interactive system, it is essential that the user receive fairly quick response from the system whenever he does get around to issuing a command.

c) In contrast to a transaction processing system, where a common program is shared among all users, with each user having a very small area of private memory (perhaps enough to hold a few lines of text), a general-purpose timesharing system must provide each user with enough memory to hold a complete program and related data. This was a serious problem in the days of high-cost memory.

D. Today, of course, the need to make efficient use of an expensive resource (the CPU) is no longer an issue, and for this reason many of the timeshared systems of the 1980's and 1990's are now only of historical interest.

However, one system that began as a timeshared system has played a very influential role: Unix.

1. Unix was originally developed at Bell Telephone Labs (now part of AT&T) in 1969.

2. An important variant was developed at Berkeley starting in 1977.

3. Today, the following systems are all descendants of one or the other of these:

a) Darwin - the core of MacIntosh OSX

b) Solaris - the system on which Java was first developed (Sun Microsystems - now part of Oracle)

c) HP/UX (Hewlett-Packard)

d) AIX (IBM)

4. Though Linux is technically a separate system, many of its core ideas come from Unix. Due to trademark issues, one cannot refer to it as “Unix”, but for all intents and purposes that is really what it is!

5. Ideas from Unix also influenced MS/DOS, which became the foundation on which Windows was first developed.

6. Unix has also played a central role in Operating Systems research in CS.

VII. **Personal Computers**

A. Operating systems evolved in the context of fairly expensive large computers, with efficient utilization of resources being a high priority.

B. The personal computer has its roots in “hobbyist” systems developed in the 1970’s, followed by the IBM PC (1981) and Macintosh (1984). Because PC’s are based on inexpensive technology, efficient utilization is not a key issue - indeed, the average PC spends most of its time sitting idle.

C. Early PCs ran a variety of one-user operating systems, but by the late 1980’s two had become widespread:

1. MS/DOS (Microsoft) running on Intel x86 platforms

2. Macintosh OS running on Macintoshes which at that time used either the Motorola 68000 or Power PC processors.

3. Both systems resembled  a cross between one user and stacked job batch systems, with a small IO library in memory without any protection and an interactive command interpreter reading commands typed by the user (MS/DOS) or a GUI (Macintosh) in place of input coming from a card reader or tape.

4. In contrast with stacked job batch systems, however, neither system had any provisions for protecting the memory belonging to the operating system or command interpreter from being damaged by an error in the program being run.

D. In the late 1980's MS/DOS morphed into Windows, using a GUI instead of a command interpreter as the primary mechanism for user interaction, and both began to support multiprogramming.

1. Of course, the motivation for multiprogramming in PC's is very different from efficient uses of expensive resources that was the original motivation.  Reliance on multiprogramming has become part of how we work.

   For example, as I am writing this lecture on a word-processor, I am also running an email program and periodically use my web browser to check various facts.

2. Early versions of Windows and multiprogrammed versions of Mac OS did not provide for memory protection, and used cooperative rather than preemptive multitasking.  (In cooperative multitasking, a job is not forced to yield the CPU, which allows a CPU-intensive program or one with an infinite loop to "hog" the system.)

   a) These capabilities were added to later versions of Windows.

b) In 2001, Apple replaced its earlier “Classic” MacOS with a totally new OS (OSX) whose kernel (Darwin) was a descendant of Berkeley Unix) and therefore supports both memory protection and preemptive multitasking.

E. Linux emerged as a third alternative in the 1990’s.

1. With its Unix heritage, Linux has always incorporated many of the ideas developed in conjunction with multiprogramming and timeshared systems (such as memory protection and pre-emptive multitasking).

2. Linux systems are often accessed using a text-based command interpreter (the shell). However, current Linux systems incorporate a number of GUI features.

a) A windowing system known as XWindows.

b) Two different GUI desktops: Gnome and KDE

## VIII. **Handheld (Mobile) Devices**

A. There are three major operating systems in widespread use in handheld devices (smart phones and tablets) today

1. iOS - used in devices manufactured by Apple

a) Originally known as iPhone OS but renamed iOS because it is also used on Apple's tablet computers (iPad and iPad mini).

b) The current version is version 8 - quite a rapid evolution for a system whose first software development kit was released by Apple in 2008!

c) At the lowest level, iOS is based on Darwin - the Unix kernel that is also at the heart of MacOSX. (Though the Unix command line is not available)

2. Android - an open-source operating system owned by Google - is used by most other smartphones and many tablet systems. It's kernel is derived from Linux - but again the Unix command-line is not available.

3. Windows 8 - used on Microsoft tablets.

B. These operating systems differ from PC operating systems (e.g. Windows, OSX, Linux) in several major ways:

1. They are designed to support touch rather than mouse and keyboard input.

2. Software comes in the form of "apps" that are often downloaded from an "App Store" - in many cases curated by the OS developer (e.g. Google Play, Apple App Store, Windows Store) though not always (e.g. Amazon App Store).

3. Operating system functions are much less visible to the user.

## IX. Embedded and Other Real-Time Systems

A. All of the systems we have been looking at so far have been designed for general-purpose computing. There is, however, another broad category of computing use called **real-time** systems.

1. A real-time system is interacting with hardware devices in real time, and is therefore characterized by inflexible time constraints: if the system cannot respond to external events within some time limit, bad things can happen.

2. One major kind of real-time systems is an embedded system that is used to control a physical system.

a) Examples

(1) Simple Cell phones (that only provide telephone-type functionality)

(2) Entertainment devices like TV set-top boxes, DVD players, game consoles ...

(3) Devices that control devices like an ATM.

(4) Automobile systems that control fuel injection, anti-lock brakes ...

(5) Systems used to control airplanes, spacecraft, missiles, etc.

(6) Systems used in the power grid to control power plants and the distribution of electricity

• • •

b) If such an embedded system cannot respond quickly enough to events in the device(s) it is controlling, negative consequences may range from user dissatisfaction to an automobile, airplane, spacecraft, or missile crash or a power blackout.

3. Another major kind of real-time systems is a system that is used to collect data from some sort of instrument (e.g. in a laboratory).

   If such a system cannot handle the data as fast as it is being produced, information may be lost.

B. Real-Time Systems can be classified as soft real-time or hard real-time.

1. In a soft real-time system, while failure to meet the time constraints is undesirable, it is not catastrophic.

   Soft real-time systems may utilize appropriate software running on a general-purpose platform (which may be dedicated to this one application or may be running other applications as well), or they may use a special operating system as on a hard real-time system.

2. In a hard real-time system, failure to meet the time constraints is unacceptable.

   Hard real-time systems use special-purpose real-time operating systems.

   a) Often, this is a "stripped-down" version of a general purpose operating system (e.g. a real-time version of Linux).

   b) There are also special systems designed from the ground up as real time systems - e.g. Windows CE, iPhone OS, Android ...

C. The system on which a real time OS runs typically comes configured with the software specific to its function, with no provision for installing new software other than updating the existing software.

## X. Today's OS Landscape

A. When one uses the word "operating system" today, what often comes to mind first is PC operating systems (e.g. Windows, MacOS, Linux), or mobile device systems (e.g. iOS, Android, Windows).

B. Network servers are also ubiquitous. These may run special server or "enterprise" versions of Windows, MacOS, or Linux.

C. Also, mainframe ("big iron") systems are still used extensively by larger corporations. (Think for a moment about the information processing needs of a bank or an insurance company, or the census, for example). IBM dominates this market, with zOS being its dominant system. These may run a transaction processing system known as CICS (Customer Information Control System) that is extensively used by people like bank tellers.

D. Real-time operating systems are found in many contexts other than computer systems.

E. Finally, in a networked world file storage services that were once exclusively provided by the operating system on a single machine are instead provided over a network. (Network file systems and cloud computing)

## XI.Summary: Key Concepts

A. In this survey of the history of operating, we saw the roots of several key ideas that play a central role in modern operating systems.

B. The need for protection if a system is serving multiple users (or multiple tasks for the same user)

C. Multiprogramming

D. The notions of preemption and priority.

E. The provision of various sorts of user interface whereby the user can specify programs to run.

1. Job Control language in batch systems (or .bat files on Windows)

2. A command line interpreter such as the shell in Unix-like systems (including MacOS) or the command line in Windows systems

3. A graphical interface such as the MacIntosh Finder, Windows Explorer, of Gnome or KDE on Linux - using either a mouse and keyboard or touch screen input.