Welcome to Operating Systems. This course is designed to introduce you to the fundamentals of operating systems. The course uses specific features of real operating systems to illustrate and reinforce the fundamental concepts.

Prerequisites include a good knowledge of data structures and computer organization.

You should also have had reasonable experience with assembler language programming for some contemporary processor (e.g. IBM S/390 or Intel X86) and strong skills in the C or C++ programming language.

C is the programming language of choice, and it is used in all code illustrations.

While not as important, a course in digital design will prove helpful.
In This Module...

- Basic material about the course
- The study of operating systems
- The OS API and the user interface
- Processor modes
- Computer and operating system history

Programming Assignments

Programming assignments are used to reinforce the concepts you learn in this course. There are three types of assignments possible:

- illustration of various system calls for one or more real, operating systems;
- implementation of OS algorithms and simulation of their effect; and
- implementation of OS algorithms in the context of a “toy” operating system, or a real operating system with readily available source code (e.g. Linux or FreeBSD).

Additional details about each of the programming assignments are provided on the class web pages.
Quizzes and Final Examination

- In addition to the programming assignments, 6 quizzes and a final examination are used to assess your mastery of fundamental operating system concepts.

- To focus your attention on the material to be covered, the set of all questions that may be used on the quizzes and the final examination is provided on the class web page—without identification of the correct answers; you must determine those yourself!

Textbooks

- The textbook used in this class is **Operating Systems Design and Implementation** (third edition), by Tanenbaum and Woodhull (Prentice Hall, 2006).

- Other modern operating system textbooks cover basically the same fundamental topics, but you may occasionally benefit from reading material written by other authors. A few of these other books include:
  - **Operating Systems Internals and Design Principles** (sixth edition), by Stallings (Prentice Hall, 2008)
The Study of Operating Systems

There are two primary approaches to the study of operating systems:

- examine the way in which application programs can use the operating system, and
- examine the design and implementation (the algorithms, data structures, and so forth) of the operating system.

We will use both of these approaches in this course.

The Goals of an Operating System

An operating system has two major goals:

- to manage the computer system’s resources, and
- to provide the functions used by application programs for the use of complex, shared, or restricted resources.

What are these resources?

- The hardware of the computer system comprises the most obvious set of resources. This includes the central processor(s), the primary memory, and all the hardware associated with input/output. We will cover each of these in detail in this course.
- Information is another important resource. The content of every file or data set must be managed by the operating system. We will also cover this resource in detail.
There are two ways in which resources may be shared:

- **Sharing in time.** For example, a processor (CPU, core) is used exclusively by one application for some period of time, and then another application will be allowed to use the processor.

- **Sharing in space.** Memory is a good example of this type of sharing. In many systems there are multiple applications in memory at the same time, each using some part of the overall memory provided by the system.

One of the tasks related to managing the computer systems resources is to ensure orderly sharing of these resources. As we will see, this is no simple feat.

---

**Viewing an OS as a Virtual Machine**

- An easy way to view an operating system is as a virtual machine that provides a convenient way for applications to use resources.

- A particular “toy” operating system provided with a textbook has almost 3000 lines of complex C code for managing the text-mode terminal (e.g. a monitor and keyboard on a PC). Clearly we don’t want application programs to include this much code just to be able to read and write the terminal.

- The “virtual machine” looks like any other computer system, but you can think of it as having additional simplified instructions (actually functions) that allow easier use of the resources than if the bare machine was used.
Computer systems include many “layers” of hardware and software, and the operating system is just one of those layers.

You may already be aware that many systems use microprograms which essentially provide functions that correspond to the machine language instructions. The code in these functions manipulate the bare hardware to effect the actions of the machine language instruction.

The operating system, as another layer, provides extended instructions (functions) that provide easy access to and management of, the system’s resources.

Application programs form a layer on top of the operating system.

---

### A Typical Computer System

<table>
<thead>
<tr>
<th>Application Programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>compilers, games, presentation software</td>
</tr>
<tr>
<td>spreadsheets, command line interpreters, editors, ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operating System—API definitions</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Assembly language coding</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Microcode—defines the instructions for the machine</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Hardware—the “bare metal”</th>
</tr>
</thead>
</table>
The Application Programming Interface

- Application programs utilize system resources by invoking the operating system. Requests from application programs are passed to the operating system through the application programming interface (API) provided by the operating system.
- Different operating systems may have different APIs, but the functions are frequently similar.
- The API for an operating system is frequently called the set of system calls for that operating system.
- Some systems may provide multiple different APIs. Examples of these include IBM's OS/390 and Microsoft Windows.

The User Interface

- Some functions you might normally think of as belonging to the operating system are actually just application programs usually provided with the system.
- The most significant example is the application that provides you with your primary interface to the system.
- This could be a text-oriented interface (as in MVS, UNIX, or MS-DOS), or a graphical user interface (as in Microsoft Windows or Apple’s OS-X).
- These applications, regardless of their complexity, are not part of the operating system. They accept user input (by using the OS API) and carry out commands (again, by using the OS API as required).
Systems With Limited User Interfaces

- To amplify the difference between the user interface and the operating system, consider the microcontrollers that you might find in a modern automobile.
- Some of these may have a user interface (like that used in the entertainment system on the automobile), but the ignition system is not likely to have a user interface.
- The applications on the microcontrollers in automobiles still use processors and memory resources, and perform input/output operations.
- And they likely have an operating system that provides an API for those applications to use!

Input/Output Interfaces

- Input/output devices are connected to a computer through an interface called a device controller.
- Each device controller usually has associated device registers that appear as locations in a memory space accessible by the central processor.
- For example, to set a timer device it is necessary to write to three byte-sized registers, one indicating the timer mode, and the other two providing a 16-bit timer interval.
- Other systems (e.g. an IBM mainframe) may have device controllers that are accessed through a special processor called a channel.
- Separate programs (logically called channel programs) must be prepared for the channel to execute.
Processor Modes (States)

- Some resources must be accessed only by the operating system if order is to be preserved in a system.
- Consider the use of a printer connected to a multiuser system. Clearly only one user should be printing at a time.
- Processors on modern systems will usually have at least two processor modes, or states.
  - The mode in which user programs (applications) usually execute is called the user mode, or the problem state. In this mode certain instructions (e.g. those that access device registers) may not be executed.
  - In the other mode, usually called the kernel mode or the supervisor state, all instructions may be executed and all resources may be accessed.
- Some systems (e.g. DEC VAX-11) may have more than two processor states, but we will only consider two in this course.

OS and Computer System History

- A brief survey of operating system and computer system history will prove useful in understanding modern systems. When doing this we will talk about generations of computer systems. Successive generations are usually introduced when major technological advances are made.
- Generation 0 began with Charles Babbage’s design for his analytical engine to be used in calculating entries of mathematical tables. This first design for a digital computing device was conceived in the early 1820s! Additionally, Ada Lovelace (the daughter of Lord Byron), was the first “programmer” for this device. The programming language ADA was named after her.
A Working Model of Babbage's Difference Engine
A Working Model of Babbage’s Difference Engine

Half of Babbage’s Brain and a Notebook
The equipment in the first generation of practical computing hardware consisted of “hard-wired” plugboards, enormously heavy mechanical devices, and vacuum-tube implementation of circuits implementing logic (i.e. AND and OR gates).

There were no operating systems, programming languages or networks in this generation.

Punched cards were introduced in the early 1950s. Each card, the same size as a U. S. dollar bill, contained 80 bytes of data. Many devices existed to process data in this format, including keypunches (to enter data onto the cards), card sorters, and listing devices.
Perhaps the most important technological advance in this generation was the use of the transistor.

Vacuum tubes, previously used in computing devices, generate enormous amounts of heat and consume lots of energy. In contrast, transistors generate little heat and consume little energy. Since transistors don’t operate by “burning themselves up” (like light bulbs), the reliability of systems was also greatly improved.

The University of Manchester (England) had a prototype transistor-based computer operational in November 1953.

The first commercial fully-transistorized computers were produced in 1957.

Operating systems also became common in the second generation. Previously users “took over the machine” to run a program, requiring significant time to mount magnetic tapes, load programs, set switches, and debug at the operator’s console.

The concept of a job, represented by a deck of punched cards containing commands and data, significantly reduced the time required to execute a set of related programs (e.g. compile or assemble a program, link separate modules together, and then run the resulting program). A command interpreter processed the commands, and the operating system managed resources like primary and secondary storage, input/output devices, and the processor.
After the deck of cards representing a job was prepared off-line using a keypunch, it was submitted to the computer system, where it processed as a unit with little or no operator intervention. After it was completed, the next job could be immediately started.

There were two basic types of computer systems in this generation:
- Scientific machines (typified by the IBM 7094) were used, as expected, for jobs requiring large amounts of computation.
- Business machines (typified by the IBM 1401) were used for jobs that were input/output intensive, but needed relatively limited calculation.
An IBM 1401 System

IBM 026 Keypunch (Vacuum Tubes)
Generation 2: Batch Processing

- In a clever combination of scientific and business systems, batch processing was used to speed job processing and make more efficient use of the systems.

- Batches of jobs (that is, multiple sequential jobs) were first copied to magnetic tape, typically using an IBM 1401.

- The tape was then physically moved to the machine where they were to be processed, typically an IBM 7094.

- The jobs were read and processed, one after another. Reading from magnetic tape is much faster than reading from punched cards.

- Output from the processing of each job was written to magnetic tape.

- When the processing of the entire batch was finished, the magnetic tape containing the output was moved back to the IBM 1401, where the execution output was copied from tape to paper (and punched cards, as necessary).
**Generation 2: A Typical Job**

- A typical job included a combination of control cards and data (that is, commands, program source statements and data).
- Control cards could be distinguished by the presence of a special character in the first column or two. For example,
  - Univac Exec 8 used a card with rows 7 and 8 punched in the first column.
  - IBM OS/360 used a /, sometimes in the first two columns.
  - VAX/VMS used a $ in the first column.
- Control cards were used to define the beginning of a job, to request the execution of a program, or to define a collection of data to be used by a program.
- The typical programming languages in use during this generation were
  - FORTRAN – FORmula TRANslator, and
  - COBOL – COmmon Business Oriented Language.
- Operating systems included IBSYS and FMS – FORTRAN Monitor System.

**Generation 3: 1965-1980**

- The major hardware development that introduced the third generation was the integrated circuit.
- Instead of using discrete components (transistors, resistors, etc.) to build a system, well-defined logical circuits were constructed as single components. This resulted in smaller and more power-efficient systems.
- Another major architectural change united the features of scientific and business computer systems. Instead of requiring the use of two different classes of machines (scientific and business), a single system providing features of both types of machines was introduced.
- The most popular of these was the IBM System/360. Multiple models this system was produced, varying in the speed and features they provided.
- Many other manufacturers also produced similar systems.
Perhaps the most important operating system in the third generation was IBM’s OS/360.

It was designed to successfully accommodate the needs of all users on every model of the System/360 computer system. This goal was never completely achieved.

The OS/360 operating system included millions of lines of assembler code written by thousands of programmers.

It isn’t surprising, then, that the numbers of bugs in the system remained essentially constant, even over many releases.

Even so, OS/360 satisfied the needs of almost all customers.
Generation 3: New Features

- **Multiprogramming**—allowed multiplexing of execution (on the processor) with input/output operations. This required partitioning memory into regions, each of which was used for a single program.
- **Spooling**—input/output from executing programs was read from and written to disk files; these files were moved between physical devices (card readers / printers and disk). SPOOL is an acronym for **Simultaneous Peripheral Operations On Line**.
- **Timesharing**—multiple users at on-line terminals could edit, compile, and test small programs while batch programs ran in the background. Terminals could be directly connected to the computer, or users could use modems to connect remote terminals to the computer using telephone lines.

Generation 3: New Architecture

- During this generation, **DEC** (Digital Equipment Corp.) introduced the PDP-1, a system based on a new architecture.
- PDP is an acronym for **Programmed Data Processor**.
- The PDP-1 and its successors were notable for their relatively small size and cost, and enabled computer systems to be employed in many environments where larger systems would have been too costly.
- This series of machines terminated with the very popular **PDP-11**.
- Like the IBM System/360, numerous other manufacturers introduced machines similar in size and capability to the DEC computers.
- Machines like these were called **minicomputers**.
MULTICS—MULTiplexed Information and Computing Service. MIT, Bell Labs, and GE developed the idea of a computing utility (similar to a power plant) that could serve large numbers of users. MULTICS introduced many seminal ideas in system design.

UNIX® from AT&T/Bell Labs.
- Source code for early versions was licensed and widely used in university courses.
- Later versions (V7 and later) were restricted.
- Other similar systems (notably FreeBSD, Linux, and MINIX) are freely available in source code form.
- The POSIX standard for UNIX-like systems was developed by IEEE.

Generation 4: 1980...

- The development of large scale integrated circuits (LSI) containing thousands of transistors and other components enabled the development of complete processors in a single circuit package.
- This was perhaps the most significant event in the fourth generation of computers.
- These microprocessor devices revolutionized computing, especially as additional development led to very large scale integrated circuits (VLSI) using complementary metal oxide semiconductor (CMOS) technology containing hundreds of thousands, even millions, of transistors.
- Families of processors have been produced, starting with small 4-bit systems.
- 64-bit microprocessor systems are common today.
Generation 4: New Peripherals

- The fourth generation saw the introduction of new low-cost peripherals. Examples:
  - graphic displays
  - graphical input devices, like mice
  - secondary storage (disk) devices that are physically smaller, faster, and with significantly more capacity than earlier devices, including removable media devices like floppy disks.
- Another “device” with enormous impact in this generation is the network, which enables communication between computer systems.
- Networks were originally closely tied to computer system manufacturers. So IBMs could talk to IBMs, and DECs could talk to DECs, but for IBMs and DECs to communicate was difficult.
- The boom of networking was due to the development of TCP/IP. This protocol family, and the explosive growth of microprocessors and networking provided the basis for the Internet.
- Indeed, the Internet can correctly be seen as marking the beginning of a new generation of computer systems.

Generation 4: New Operating Systems

- Widespread use of microprocessors spurred the development of new operating systems. Some of the more notable of these include
  - MS-DOS
  - Microsoft Windows
  - Apple OS-X
- UNIX is still popular in the fourth generation.
- High-end workstations and supercomputers almost always use UNIX, and Linux (with essentially the same features as UNIX) is enormously popular. It runs on numerous architectures, including mainframes, microcomputers and embedded systems.
- IBM’s z/OS is one of the current operating systems for mainframes. It is backward compatible to systems dating back to OS/360.
In the next module we’ll look at two major operating system concepts:

- processes,
- files/datasets.

Various system calls for managing processes will be examined.

Programs using selected system calls will be examined.