Instructor:	Dr. Jim Plusquellic
Lectures:	T-Th 2:30-3:45, in SS409
Office Hours:	T-Th 3:45-5:00, ECS 212
Text:	Operating Systems: Design and Implementation, by Andrew S. Tannenbaum and Albert S. Woodhull
Supplementary text:	Operating Systems Concepts, by Abraham Silberschatz and Peter B. Galvin
TA:	Mr. Hua Yuan, yuan@umbc.edu
Further Info:	http://www.cs.umbc.edu/~plusquel/

1.1 What is an Operating System (OS) ?

An Operating System is a special system program that acts as an intermediary between the user of a computer and the computer hardware.

User 1 | User 2 | User 3 | User 4 ... User n

Applications	→	Word Processor/Web Browser/etc.
System programs	→	Compiler/Command Interpreter/etc.
Operating System	→	Device Drivers/Memory Mgmt/etc.
Hardware	→	CPUs/memory and I/O devices

Primary Goal: Provide convenience to the user:

Provide user with an abstraction or simplified interface to the hardware.

Provide user with virtual machine that is easier to program than the underlying hardware.

Secondary Goal: Make use of computer resources in an efficient manner.

Provide an orderly and controlled allocation of resources, e.g. CPUs, memory, I/O devices, files, databases, etc., among the various programs competing for them.

Note: These two goals are sometimes contradictory.

e.g. Is providing the user with the option of printing directly to the printer convenient -> 'copy text.dat LPT1:' ? Is it orderly and controlled?

Operating System Views

As a control program:

Control user programs to prevent improper use of resources.

As a resource allocator:

Resolves conflicts over computer resources efficiently and fairly.

Identifying characteristic of an Operating System

The one program (kernel) running at all times on the computer (usually in supervisor mode) with all else being applications (running in user mode).

1.2 Operating System History

First Generation:

- Programs were written in machine language.
- No Operating System.

Second Generation:

- Programmers submitted Fortran programs on card decks.
- **Off-line** job preparation/printing introduced.
  - Multiple **jobs** were written to magnetic tape.
  - Job output was printed from magnetic tape.
  - Card readers and printers were NOT connected to main CPU.
- **Batch systems** introduced in which jobs (Fortran programs) were read from magnetic tape by a primitive Operating System (resident monitor) and ran to completion.

- Advantages:
  - Faster magnetic tapes increased CPU utilization.
  - Job preparation or **set-up** time reduced because jobs with similar requirements batched together.
  - Multiple card readers and printers used to read and write magnetic tape.

- Disadvantages:
  - Increased **turnaround** time for the programmers.

Third Generation:

- **Random-access** disks:
  - Improved off-line preparation of jobs/printing of output when compared with **sequential-access** tapes because card reader/printer and CPU could access data on disk concurrently.
  - Made **job scheduling** possible because *pool* of jobs which were ready to run on disk could be accessed the OS.

- **Spooling** (Simultaneous Peripheral Operation On Line) introduced. Spooling overlaps the I/O of one job with the computation of other jobs.
  - Increased CPU and I/O device utilization.

- **Multiprogramming** introduced to further increase CPU utilization.
  - Problem addressed: Speed of mechanical I/O devices much slower than electronic devices. e.g. the CPU.
  - Multiple jobs kept in separate memory partitions.
  - Operating System switches between them when running job or **process** blocked for I/O.
  - Memory protection hardware introduced to prevent one process from interfering with another.

Job scheduling: OS determines which of several jobs on disk to load into memory.

CPU scheduling: OS selects a job to run from several jobs ready to run in memory.

- **Timesharing** or **multitasking** introduced to improve system response time for users.
  - A variant of multiprogramming in which each user has an **on line** terminal.
  - Time slices prevent one process from over-utilizing the CPU.

Fourth Generation:

- LSI paved way for inexpensive Personal Computers.
- Operating Systems include MSDOS, Windows, Windows95, Windows NT, Unix (Linux).

- **Network Operating Systems**:
  - Stand-alone machines each running their own operating systems.
  - Users have ability to log onto other machines and run programs.
- **Distributed Operating Systems**:
  - Single Operating System running on several machines.
  - Users are not aware of which machine is actually running their program.

1.3.1 Processes

A **process** is a program in execution.

A process consists of:

- An address space containing the executable program, the program's data and its stack.
- A set of registers (PC, stack pointer, etc).
- Open files and file pointers
- A unique process identifier (**pid**).

A process's **state** is saved in an OS table called a **process table**.

Processes can create **child** processes (in UNIX, through the fork() system call).

Processes communicate and synchronize through **interprocess communication** mechanisms such as pipes, messages, shared memory or signals.

(Processes will be covered in more detail in Chapter 2).

1.3.2 Files

OS hides details of disks by providing a device independent file system abstraction composed of files and directories.

UNIX and DOS file systems are hierarchical.

UNIX OSs additionally provide:

- A protection code mechanism for each file and directory.
- The ability to **mount** other devices, e.g. floppy disks, onto the **root file system**.
- **Special files** that make I/O devices look like files, e.g. /dev/hda1
  - **Block special files** used to model devices that consist of a collection of randomly addressable blocks.
  - **Character special files** used to model devices that accept or output a character stream.

DOS special files include LPT1:, COM1, etc.

A **pipe** is a pseudofile that can be used to connect two processes together.

1.4 System Calls

System calls define the interface between the user application program and the operations provided by the OS.

System calls define the capabilities of the OS.

An example is READ. The C library function *wrapper* **read** is defined as:

```
count = read(file, buffer, nbytes);
```

which takes three parameters, the file, a pointer to a buffer in which to place the data and the number of bytes to read.

read returns the number of bytes successfully read or -1 for an error.

The system call itself, READ, is called from within the library routine in a way similar to a regular procedure call except:

- Part of the process state--registers, PC, SP, etc.--is saved
- A *trap* instruction (or some means of switching from user to kernel mode) is used to invoke the READ routine.
- The READ system call checks the parameters passed (possibly verifying the request against other process or user info) and, if valid, performs the requested action.
- The READ system call then returns control back to the user process.

Other C system calls, e.g. **fread**, are complex library routines that provide extended functionality beyond that provided by the OS system call interface.

Minix has system calls (described on pages 22-36) for:

- Process management, e.g. **fork**()
- Signals, e.g. **alarm**(...)
- File Management, **open**(...)
- Directory and File System Management, e.g. **mkdir**(...)
- Protection, e.g. **chmod**(...)
- Time Management, e.g. **time**(...)

1.5 Operating System Structure

The system call interface defines what the OS looks like on the outside.

OS internal organization can be classified into four categories:

- Monolithic
- Layered
- Virtual machines
- Client-Server model

Monolithic Systems

- Characterized as a collection of procedures and interrupt service routines that have no modular structure.
- Each procedure has a well-defined interface and is free to call any other routine.
- Most commercial OSs are more or less monolithic.

Layered Systems

An attempt to impose some order among the set of OS procedures.

The most straightforward breakdown of tasks consists of a three layered system:

- A **main** procedure that is responsible for invoking the requested service procedure. For example, this routine may perform the following:
  - Push the system call parameters onto the stack or move them into registers.
  - Execute the special trap instruction to switch to kernel mode.
  - Determine service routine to invoke.
- A collection of service procedures to carry out the system calls.
- A collection of utility procedures that, for example, check the validity of the parameters and fetch user data.

More complex layered systems include **Multics** and the **THE** system.

Multics is organized as a set of concentric rings where:

- The inner rings are more privileged than the outer rings.
- Procedures in the outer rings call procedures within the inner rings with a trap instruction (a generalization of system call).
- The whole OS is part of the address space of every process.
  - Hardware mechanisms make it possible to designate individual procedures (memory segments) as protected.

Virtual Machines
Introduced by IBM. Their OS is called VM/370.

System consists of:

- The VM/370 OS which runs on the bare hardware.
  - Handles multiprogramming.
  - Provides several **virtual** machines to the next level up which are exact copies of the bare hardware.

- Virtual machines each run their own OS (which are emulated by VM/370).
  - User program system calls are trapped to the OS running on the virtual machine.
  - OS then issues hardware I/O instructions.
  - These I/O instructions are trapped by the VM/370 and carried out on behalf of the OS that invokes them.

This completely separates multiprogramming (done at a lower level to allow multiple virtual machines) from the issue of providing a convenient interface to the hardware.

Commonly used today on Intel 80386, 80486 and Pentiums running MS Windows.

- Intel provides a virtual 8086 mode.
- Multiple virtual 8086 sessions can be created to run MS-DOS programs.

The Intel/Microsoft virtual machine variant is only able to emulate an 8086 while the VM/370 can, in fact, emulate another VM/370 in one of its virtual machines.

The Intel/Microsoft virtual machine provides an exact copy of a different machine.

Client-Server Model

Continuation of trend to move system functions up to the user level and minimize the size of the kernel.

- Most OS system functions implemented as **server processes**.
- User programs (**clients processes**) request system services by sending messages to server processes.
- Kernel only responsible for handling communication between clients and server processes and scheduling processes.

OS decomposed into a set of servers providing, for example:

- File service
- Terminal (tty) service
- Process service
- Memory service

Advantages include:

- Simplicity
- Reliability
- Modularity
- Application to distributed systems

The main disadvantage is performance; there is usually more overhead for a message than a procedure call.

The Mach micro-kernel is an example of such a system.