

A computer system consists of:

- **Hardware**

- Physical devices
 - * Processors
 - * Memory
 - * I/O devices
- Microprogramming
- Machine language (instruction set)

- **Software**

- System programs
 - * Operating system
 - * Compiler
 - * Editor
 - * Command interpreter
- Application programs
 - * Banking system
 - * Airline reservation
 - * Games

What is an operating system?

Two views:

1. Top-down view: Extended machine

- **Covers the details of the hardware**
- **Provides more friendly interface to programmers**

2. Bottom-up view: Resource manager

Consists of four parts:

- **Processor manager**
- **Memory manager**
- **I/O device manager**
- **File manager**

Types of operating systems:

- **Batch systems:**
Run jobs one by one to completion. Long turnaround time.
- **Interactive systems:**
Short turnaround time. Each job uses part of CPU time (quantum) without waiting for others to finish.
- **Real-time systems:**
The system must respond within a fixed amount of time to ensure correct performance. Used in time critical environments.
- **Hybrid systems:**
Batch system (background) + Interactive system (foreground).
- **Distributed systems:**
Distributed computation among several physical processors. The processors do not share memory or a clock. They communicate with each other through communication links.

- **Multiprogramming:**

Partition memory into several segments with a different job in each partition. Overlap CPU job and I/O job to increase CPU utilization.

- **Time-sharing:**

A variant of multiprogramming.

- **Difference between batch multiprogramming vs. time-sharing:**

	Batch	Time-sharing
Obj.	Maximize CPU use	Minimize response time
Instr.	Job control language	Command from terminal

- **Network O.S. vs. distributed O.S.**
 - **Network O.S.**
 - * users are aware of the existence of multiple computers
 - * can login to remote machines and copy files from one machine to another
 - * each machine runs its own local O.S. and has its own users
 - **Distributed O.S.**
 - * appears to users as a single computer system.
 - * users should not be aware of where their programs are being run or where their files are located.
 - * O.S. handles them automatically and efficiently.

Review:

- **Components of a computer system**
- **Definition of O.S.: extended machine + resource manager**
- **Multiprogramming**
- **Difference between batch multiprogramming and time-sharing system**
- **Difference between network O.S. and distributed O.S.**

Operating System Concepts (Unix system based)

Kernel: The kernel is the O.S. It provides services such as

- file system
- memory management
- CPU scheduling
- I/O

Program: an executable file.

Process: an instance of a program that is executed by O.S.

Process ID: a unique number for each process.

Process table: a data structure within the kernel that contains one entry per process including process ID, process state, program counter, CPU registers, memory limits, list of open files, etc.

Child process: a process created by other process in Unix.

Process tree structure.

System calls: a set of “extended instructions” provided by O.S., providing the interface between a process and the O.S.

Example: Read a certain number of bytes from a file.

```
count = read(fd, buffer, nbytes)
```

File: a collection of data.

File types:

- Regular file (data bytes)
- Directory (the name of other files)
- Special file (I/O devices: character, block)

Directory tree.

File access bits.

File descriptor: a small integer used to identify an open file.

Pipes: a pseudo file that can be used to connect two processes.

Shell: Unix command interpreter.

Examples of shell commands:

date

date > file (output redirection)

sort < file (input redirection)

sort < file1 > file2 (input + output redirection)

cat file1 file2 | sort > file3 (pipe + output redirection)

Unix system calls:

“Extended machine instructions.” O.S. provides services to users through system calls.

Used for different purposes:

- System calls for process management
 - `fork()`: the only way to create a new process in Unix. Create a copy of the process executing it.
fork returns 0 in the child, returns child's pid in the parent.
 - `exit(status)`
A process terminates by calling exit system call.
status: 0-255, 0: normal, others: abnormal terminations.
 - `waitpid(pid, status, opts)`
pid: specific child, -1: first child.
status: child exit status.
opts: block or not.

- **execve**

The only way a program is executed in Unix.

`s = execve(file, argv, envp)`

- **Example: A simplified shell.**

- **System calls for signals.**
Signals are called “software interrupts.” One process can send signals to another process.
 - Signal names: SIGINT, SIGKILL, SIGALRM, ...
 - What to do with a signal system call?
 - * Catch a signal: `signal(sig, func)`.
Provide a function that is called whenever a specific type of signal occurs.
Need to re-enable signal catching.
 - * Ignore a signal: `signal(sig, SIG_IGN)`.
All signal, other than SIGKILL can be ignored.
 - * Allow the default to happen: `signal(sig, SIG_DEF)`.
Normally, a process is terminated when receive a signal.
 - Send a signal: `kill(pid, sig)`.

- **System calls for files**

- read, write
- create, open and close a file:
fd = creat(filename, mode)
fd = open(file, how)
close(fd)
- random access a file:
pos = lseek(fd, offset, whence)
- duplicate the file descriptor:
fd2 = dup(fd)
fd2=dup2(fd, fd2)
- create a pipe:
pipe(&fd[0])
returns two file descriptors:
fd[0] : for reading
fd[1]: for writing
- example for using pipe system call.

Review:

- **Operating system concepts:**
kernel, program, process, system calls, file, directory, file protection, pipe and shell.

- **Unix system calls:**
fork, exit, waitpid.

Review:

- **Unix system calls:**
execve, signal, kill, read, write, lseek, creat,
open, close, dup, pipe.
- **Examples of using system calls:**
Shell and pipe.

Operating System Structure

- **Monolithic systems:**
 - O.S. consists of a collection of procedures.
 - All procedures linked together.
 - There is no structure.
 - Interface between the user program and O.S.: kernel call (a trap instruction).
 - General structure model (three levels):
 - * main program: call the service procedure
 - * service procedures: execute the systems calls
 - * utility procedures: some commonly called functions

- **Layered systems:**
 - Organize the O.S. into a hierarchy of layers.
 - Example: the THE O.S. system by E.W. Dijkstra (six layers).
 - * **Layer 0:**
Processor management: allocate processor, provide multiprogramming.
 - * **Layer 1:**
Memory management: allocate memory for processes, paging.
 - * **Layer 2:**
Handle communications between processes and console.
 - * **Layer 3:**
I/O device management.
 - * **Layer 4:**
User programs.
 - * **Layer 5:**
System operator process.

- **Virtual machines**

- **VM: Virtual machine monitor.** Does multiprogramming, providing several virtual machines (VM), each of which is identical to the hardware machine and can run different O.S.
- **CMS: conversational monitor system** (a single user O.S.).
- **User process accesses I/O:**
 - * executes a system call (trap to its own virtual machine).
 - * CMS issues the hardware I/O instruction, trapped to VM.
 - * VM simulates the hardware.

- **Client-server model**

- **Micro kernel.**
- **Most of O.S. functions performed at user level.**
- **User process (client) sends a request to server process.**
- **Kernel handles communication between processes.**
- **Allow O.S. to be distributed, more reliable.**

Review:

- **Internal structure of O.S.**
 - **Monolithic systems**
 - **Layered systems**
 - **Virtual machines**
 - **Client-server model**