Overview of Operating Systems

Instructor: Dr. Tongping Liu

Thank Dr. Dakai Zhu and Dr. Palden Lama for providing their slides.

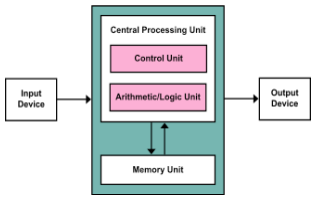
Lecture Outline

- Operating System: what is it?
- Evolution of Computer Systems and OS Concepts
- Different types/variations of Systems/OS
  - Parallel/distributed/real-time/embedded OS etc.
- OS as a resource manager
  - How does OS provide service? – interrupt/system calls
- OS Structures and basic components
  - Process/memory/IO device managers

Lecture Outline

- Operating System: what is it?
- Evolution of Computer Systems and OS Concepts
- Different types/variations of Systems/OS
  - Parallel/distributed/real-time/embedded OS etc.
- OS as a resource manager
  - How does OS provide service? – interrupt/system calls
- OS Structures and basic components
  - Process/memory/IO device managers

Von Neumann Architecture



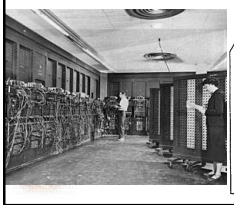
This describes a design architecture:

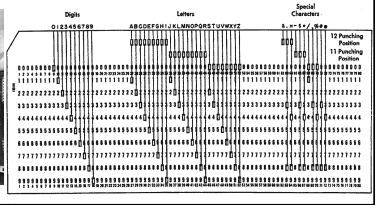
- A processing unit containing an ALU and [illegible]
- A control unit containing an instruction register and program counter
- **A memory to store both data and instructions (Stored-program computer)**
- External mass storage, and input and output mechanisms.

Before, computers have fixed uses

First Computer: ENIAC in 1940s

- Big: 27 tons, 680 ft^2, and use 150kW
- Slow: tens instructions/s
- Limited functions: addition / multiplication
- Hard to use: Button switch or Punch card I/O





First Computer: ENIAC (cont.)

- Got problem with the program?! → you are in trouble ☺
- Process one job at a time
  - Human is slow
  - CPU time is precious
- Batch systems
  - Read in more jobs
  - Process one by one
  - I/O devices are still slow?

Debug your program!

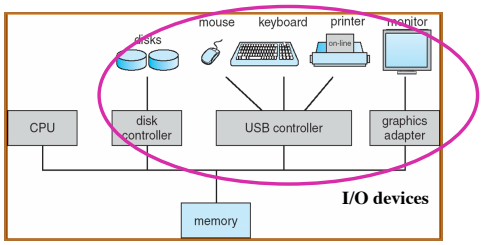
Desktop Systems: 1980s

- *Personal computers* **dedicated to a single user**;
- *Objective:* User convenience and responsiveness.
  - Individuals have sole use of computers
  - A single user may not need advanced features of mainframe OS (maximize utilization, protection).
- I/O devices – display, keyboard, mouse and printers
- Today, may run several different types of operating systems (Windows, MacOS, Linux)

Parallel High-Performance Systems

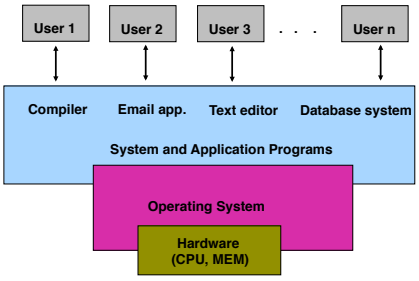
- Goals:
  - Increased performance/throughput
  - Increased reliability: fault tolerance
- Multiprocessor systems: more than one CPUs
  - *Tightly coupled system* – processors share memory, bus, IO, and a clock; communication usually takes place through the shared memory
- *Symmetric multiprocessing (SMP) vs. asymmetric*
  - SMP: each processor runs an identical copy of the operating system; all processors are peers
  - Asymmetric: master-slave

Modern Computer Hardware Organization



von Neumann Architecture: store and computation

Big Picture of Computer Systems



Operating Systems: Why do you need it?

$300

\+

OS (Windows)

$200

Do I have to?!

- Yes: you have to have an OS to use a modern computer
  - Otherwise, a set of silicon circuits doing nothing good for you !
- OS provides user friendly ***interface*** for using computers

Operating Systems: Definitions

- A program that acts as an ***intermediary*** between the user/programs of computer and computer hardware.
- From system view
  - Kernel – the program running at all times (all else being application programs)
  - Control program – controls the execution of user programs and operations of I/O devices
  - **Resource allocator** – manages and allocates *resources*
- **Goals** of Operating system
  - **Convenience**: Make the computer convenient to use.
  - **Efficiency**: Manage system resources in an efficient manner

Lecture Outline

- Operating System: what is it?
- Evolution of Computer Systems and OS Concepts
- Different types/variations of Systems/OS
  - Parallel/distributed/real-time/embedded OS etc.
- OS as a resource manager
  - How does OS provide service? – interrupt/system calls
- OS Structures and basic components
  - Process/memory/IO device managers

Multiprogrammed Systems

A single program cannot keep busy for CPU and IO, thus wasting resources

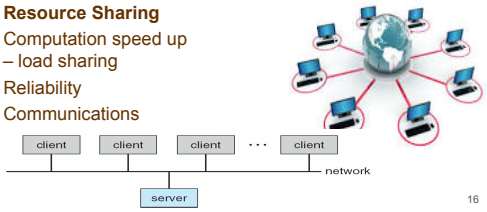
- Several jobs run "*concurrently*"
  - Job: computing → input → computing → … → output
  - *Take turns* to use CPU and I/O devices
- But ***which*** job uses ***what*** and ***when***?
  - Need a manager/supervisor → OS
- ***How*** to use the hardware (e.g., I/O devices)?
  - Resource manager/interface → OS

Time Sharing Systems

- Extension to multi-programmed systems
- Multiple **interactive** users
  - Allow on-line interaction with users;
  - ***Response time*** for each user should be short
- CPU is multiplexed among several jobs of several users that are kept in memory
  - CPU is allocated to jobs in **Round-Robin** manner
  - All active users must have a ***fair*** share of the CPU time: e.g. with 10 ms time quantum
- Example systems: IBM 704 and 7090

Distributed Systems

- *Loosely coupled system* – each processor has its own local memory; processors communicate with one another through various **communications** lines
- Advantages of distributed systems.
  - **Resource Sharing**
  - Computation speed up – load sharing
  - Reliability
  - Communications



Peer-to-Peer Computing Systems

- Another model of distributed system
- P2P does not distinguish clients and servers
  - Instead all nodes are considered peers
  - May each act as client, server or both
  - Node must join P2P network
    - ✓ Registers its service with central lookup service on network, or
    - ✓ Broadcast request for service and respond to requests for service via **discovery protocol**
  - Examples include *Napster* and *Gnutella*

Special Purpose Systems

- A **real-time** system is used when strict time requirements have been placed on the operation of a processor or the flow of data
  - *Hard* real-time: critical tasks must be completed on time
  - *Soft* real-time: no absolute timing guarantees (e.g. "best-effort scheduling"); multimedia applications;
- An **embedded** system is a component of a more complex system
  - Control of a nuclear plant or Missile guidance
  - Control of home and car appliances (microwave oven, DVD players, car engines, …)
- Example: VxWorks and eCos; Android and iOS

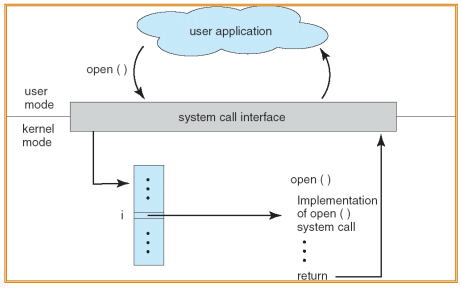
OS Interface: APIs and System Calls

- For application programmers
- ***Application programming interface (API)***
  - The run-time support system (run-time libraries) provides a *system-call interface*, that intercepts function calls in the API and invokes the necessary system call within the operating system
- **System calls** provide the interface between a running program and the operating system.
  - Generally available in routines written in C and C++
  - Certain low-level tasks may have to be written using assembly language

Lecture Outline

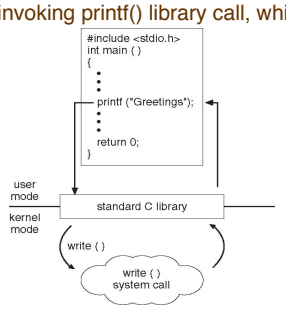
- Operating System: what is it?
- Evolution of Computer Systems and OS Concepts
- Different types/variations of Systems/OS
  - Parallel/distributed/real-time/embedded OS etc.
- OS as a resource manager
  - How does OS provide service? – interrupt/system calls
- OS Structures and basic components
  - Process/memory/IO device managers

Example: System-Call Processing



Standard C Library Example

- C program invoking printf() library call, which calls write() system call



Types of System Calls

- Process control
- File management
- Device management
- Information maintenance
- Communications
- Protection

Examples: Major System Calls in Unix

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

File management

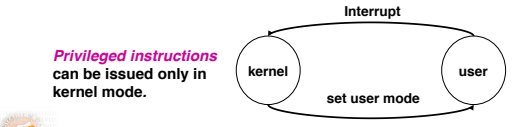
Call	Description
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Examples of Windows and Unix System Calls

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

Operation Modes

- Hardware support (**mode bit**) to differentiate between at least two modes of operations
  - ***User mode*** – execution done on behalf of a user
  - ***Kernel mode*** (also *monitor mode* or *system mode or privileged mode*) – executing on behalf of operating system
- E.g., **interrupts**: → switches to monitor mode.



Privileged instructions **can be issued only in kernel mode.**

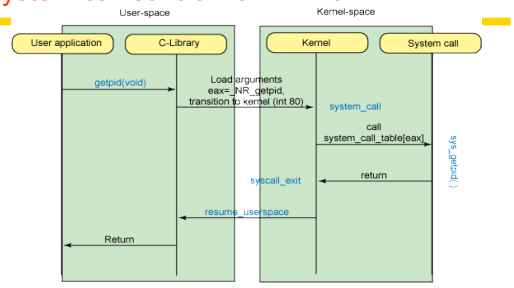
System call control flow - Linux

- User application calls a user-level library routine (`gettimeofday()`, `read()`, `exec()`, etc.)
- Invokes system call through stub, which specifies the system call number. From `unistd.h`:

```
#define __NR_getpid 172
__SYSCALL(__NR_getpid, sys_getpid)
```

- This generally causes an interrupt, trapping to kernel
- Kernel looks up system call number in syscall table, calls appropriate function
- Function executes and returns to interrupt handler, which returns the result to the userspace process

System call control flow - Linux



The idea is still the same, although code is different

Interrupt Mechanisms

- Save the current "process state"
- Interrupt transfers control to the interrupt service routine (ISR) generally through ***interrupt vector*** containing the addresses of all the service routines.
- **ISR**: Separate segments of code determine what action should be taken for each type of interrupt.
- Once the interrupt has been serviced by the ISR, the control is returned to the interrupted program.

Basic Interrupt Processing

1. The interrupt is issued
2. Processor finishes execution of current instruction
3. Processor signals acknowledgement of interrupt
4. Processor pushes PSW(*Program Status Word)* and PC to control stack
5. Processor loads new PC value through the interrupt vector
6. ISR saves remainder of the process state information
7. ISR executes
8. ISR restores process state information
9. Old PSW and PC values are restored from the control stack

What if another interrupt occurs during interrupt processing?

Classes of Interrupts

- **I/O Interrupts**: Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions
- **Timer Interrupts**: Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis, like scheduling
- **Hardware Failure Interrupts**: Generated by a failure (e.g. power failure or memory parity error).
- **Traps (Software Interrupts):** Generated by some condition that occurs as a result of an instruction execution
  - User request for an operating system service (e.g., system calls)
  - Runtime errors

Lecture Outline

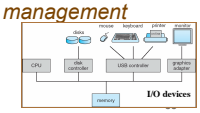
- Operating System: what is it?
- Evolution of Computer Systems and OS Concepts
- Different types/variations of Systems/OS
  - Parallel/distributed/real-time/embedded OS etc.
- OS as a resource manager
  - How does OS provide service? – interrupt/system calls
- OS Structures and basic components
  - Process/memory/IO device managers

Components in Operating System

- ***Process/thread Management***
  - ***CPU (processors): most precious resource***
- ***Memory Management***
  - ***Main memory***
- File Management → data /program
- Secondary-Storage Management → disk
- I/O System Management → I/O devices

I/O

- *Protection and Security → access management*



Process Management

- A ***process*** is a *program* in execution (active),
  - Dynamic concept, represented by ***process control block***
- A process needs resources: execution environment
  - including CPU time, registers, memory, files, and I/O devices to accomplish its task
- OS provides mechanism to
  - Create/delete processes
  - Run/Suspend/resume processes (scheduling/signal)
  - Process communication and synchronization
  - Deadlock handling

Main Memory Management

- The main memory is
  - a large array of words/bytes, each with its own address
  - a volatile storage device: content lost when power off
- The operating system will
  - Keep track of which parts of memory are currently being used and by whom
  - Decide which processes to load when memory becomes available
  - Allocate and de-allocate memory space as needed

File Management

- A file is a collection of related information (logic unit)
  - Format is defined by its creator.
- Represent programs (source/object forms) and data
- Operating system responsibilities
  - File creation and deletion
  - Directory creation and deletion
  - Support of primitives for manipulating files and directories
  - Mapping files onto secondary storage
  - File backup on stable (non-volatile) storage media

Secondary-Storage Management

- The *secondary storage* backs up main memory and provides additional storage.
- Most common secondary storage type: disks
- The operating system is responsible for
  - Free space management
  - Storage allocation
  - Disk scheduling

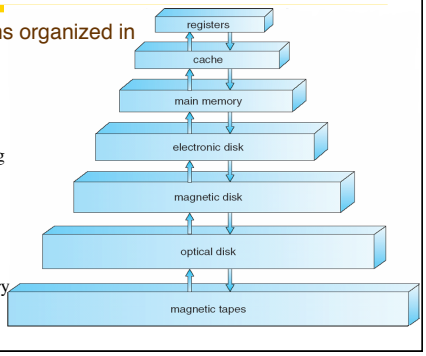
I/O System Management

- The Operating System will hide the peculiarities of specific I/O hardware from the user.
- In Unix, the I/O subsystem consists of:
  - A buffering, caching and spooling system
  - A general device-driver interface
  - Drivers for specific hardware devices
- **Interrupt handlers** and **device drivers** are crucial in the design of efficient I/O subsystems

Storage Hierarchy

- Storage systems organized in hierarchy
  - Speed
  - Cost
  - Volatility

Caching – copying information into faster storage system; main memory can be viewed as a last *cache* for secondary storage

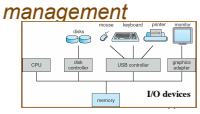


Caching

- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- Information in use copied from slower to faster storage temporarily
- Faster storage (cache) checked first to determine if information is there
  - If it is, information used directly from the cache (fast)
  - If not, data copied to cache and used there
- Cache smaller than storage being cached
  - Cache management important design problem
  - Cache size and replacement policy

Components in Operating System

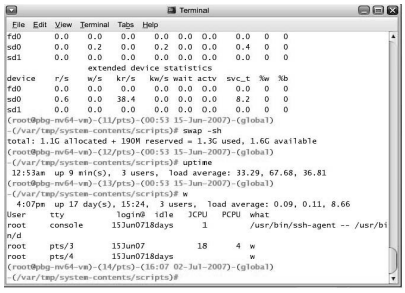
- ***Process/thread Management***
  - ***CPU (processors): most precious resource***
- ***Memory Management***
  - ***Main memory***
- File Management → data /program
- Secondary-Storage Management → disk
- I/O System Management → I/O devices
- *Protection and Security → access management*



OS Interface: Shell and GUI

- For both ***programmers*** and ***end-users***
- Two main approaches for both
  - *Command-line interpreter* (or *shell*)
  - *Graphical User Interfaces (GUI)*
- The shell
  - allows users to directly enter commands that are to be performed by the operating system
  - Is usually a system program (not part of the kernel)
- GUI allows a mouse-based window-and-menu system: click-and-play
- Some systems allow both (e.g. X-Windows in Unix)

Bourne Shell Command Interpreter



The Mac OS X GUI