CS370 Operating Systems
Colorado State University
Yashwant K Malaiya
Spring 2021 Lecture 3

Slides based on
• Text by Silberschatz, Galvin, Gagne
• Various sources
Perspective

Differences among Subroutines/traps/Interrupt service routines

- Subroutines: program specifies transfer of control
- Traps: transfer of control to a system routine
- Interrupt: hardware request transfers control to the interrupt service routine

Interrupts: Why? How?

- *Interrupt request line* is hardware
- Interrupt causes transfer of control to *Interrupt Service Routine*
- Hence need to save context. Context restored when returning.
FAQ: DMA, Driver vs Controller

When is Direct Memory Access (DMA) needed:
• When a block of data needs to be transferred memory <-> ext device (disk or network controller)

Block transfer using DMA Controller vs CPU
• CPU needs to fetch instructions for each word transfer: too much overhead
• DMA Controller, once initialized, doesn’t need to fetch instructions
• DMA: direct connection between memory and IO device

Device Driver (software) vs Device controller (hardware):
• Device controller understands software commands to handle hardware actions
• Device driver: hides device details from kernel
Multiprogramming, Multitasking, Multiprocessing

- **Multiprogramming**: multiple program under execution at the same time, switching programs when needed (older term)
- **Timesharing (multitasking)**: sharing a CPU among multiple users using time slicing (older term). *Multitasking among people* ...
- **Multiprocessing**: multiple processors in the system running in parallel.
- **Program vs process**: a program under execution can constitute one or more processes.
- **Job**: (older, somewhat ambiguous term) an executing program or a process
Today

- Multiprocessors
- OS Operations
- Storage hierarchy
- OS structures
- User interfaces
- System calls
Multiprocessors

• Past systems used a single general-purpose processor
  – Most systems have special-purpose processors as well

• **Multiprocessors** systems were once special, now are common
  – Advantages include:
    1. Increased throughput
    2. Economy of scale
    3. Increased reliability – graceful degradation or fault tolerance

  – Two types:
    1. **Asymmetric Multiprocessing** – each processor is assigned a specific task.
    2. **Symmetric Multiprocessing** – each processor performs all tasks
Symmetric Multiprocessing Architecture

Multi-chip and multicore
- Multi-chip: Systems containing all chips
  - Chassis containing multiple separate systems
- Multi-core

FAQ: How does system decide what information should be in cache?
Multiprogramming and multitasking

- **Multiprogramming** needed for efficiency
  - Single user cannot keep CPU and I/O devices busy at all times
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
  - A subset of total jobs in system is kept in memory
  - One job selected and run via **job scheduling**
  - When it has to wait (for I/O for example), OS switches to another job

- **Timesharing (multitasking)** is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating interactive computing
  - **Response time** should be < 1 second
  - Each user has at least one program executing in memory ⇒ **process**
  - If several jobs ready to run at the same time ⇒ **CPU scheduling**
  - If processes don’t fit in memory, **swapping** moves them in and out to run
  - **Virtual memory** allows execution of processes not completely in memory
Memory Layout for Multiprogrammed System

- Operating system
- Job 1
- Job 2
- Job 3
- Job 4

512M
Operating-System Operations

• “Interrupts” (hardware and software)
  – Hardware interrupt by one of the devices
  – Software interrupt (exception or trap):
    • Software error (e.g., division by zero)
    • Request for operating system service
    • Other process problems like processes modifying each other or the operating system
• **Dual-mode** operation allows OS to protect itself and other system components
  
  – **User mode** and **kernel mode**
  – **Mode bit** provided by hardware
    • Provides ability to distinguish when system is running user code or kernel code
    • Some instructions designated as **privileged**, only executable in kernel mode
    • System call changes mode to kernel, return from call resets it to user

• Increasingly CPUs support multi-mode operations
  – i.e. **virtual machine manager (VMM)** mode for guest **VMs**
Transition from User to Kernel Mode

- Ex: to prevent a process from hogging resources
  - Timer is set to interrupt the computer after some time period
  - Keep a counter that is decremented by the physical clock.
  - Operating system sets the counter (privileged instruction)
  - When counter zero generate an interrupt
  - Set up before scheduling process to regain control or terminate program that exceeds allotted time

- Ex: System calls are executed in the kernel mode
Process Management

- A process is a program in execution. It is a unit of work within the system. Program is a *passive entity*; process is an *active entity*.
- Process needs resources to accomplish its task
  - CPU, memory, I/O, files
  - Initialization data
- Process termination requires reclaim of any reusable resources
- **Single-threaded process** has one *program counter* specifying location of next instruction to execute
  - Process executes instructions sequentially, one at a time, until completion
- **Multi-threaded process** has one program counter per thread
- Typically, system has many processes (some user, some operating system), running concurrently on one or more CPUs
  - Concurrency by multiplexing the CPUs among the processes / threads

A program may involve multiple processes.
The operating system is responsible for the following activities in connection with process management:

- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for
  - process synchronization
  - process communication
  - deadlock handling

More about these later
Memory & Storage Management
Byte (B) = 8 bits (b)

Amount of info:

- A **kilobyte**, or **KB**, is $1,024$ (or $2^{10}$) bytes
- A **megabyte**, or **MB**, is $1,024^2$ (or $2^{20}$) bytes
- A **gigabyte**, or **GB**, is $1,024^3$ bytes
- A **terabyte**, or **TB**, is $1,024^4$ bytes
- A **petabyte**, or **PB**, is $1,024^5$ bytes

Measures of time

- **Milli**seconds, **micro**seconds, **nano**seconds, **pico**seconds $10^{-3}$, $10^{-6}$, $10^{-9}$, $10^{-12}$
Performance of Various Levels of Storage

<table>
<thead>
<tr>
<th>Level</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>registers</td>
<td>cache</td>
<td>main memory</td>
<td>solid state disk</td>
<td>magnetic disk</td>
</tr>
<tr>
<td>Typical size</td>
<td>&lt; 1 KB</td>
<td>&lt; 16MB</td>
<td>&lt; 64GB</td>
<td>&lt; 1 TB</td>
<td>&lt; 10 TB</td>
</tr>
<tr>
<td>Implementation technology</td>
<td>custom memory with multiple ports CMOS</td>
<td>on-chip or off-chip CMOS SRAM</td>
<td>CMOS SRAM</td>
<td>flash memory</td>
<td>magnetic disk</td>
</tr>
<tr>
<td>Access time (ns)</td>
<td>0.25 - 0.5</td>
<td>0.5 - 25</td>
<td>80 - 250</td>
<td>25,000 - 50,000</td>
<td>5,000,000</td>
</tr>
<tr>
<td>Bandwidth (MB/sec)</td>
<td>20,000 - 100,000</td>
<td>5,000 - 10,000</td>
<td>1,000 - 5,000</td>
<td>500</td>
<td>20 - 150</td>
</tr>
<tr>
<td>Managed by</td>
<td>compiler</td>
<td>hardware</td>
<td>operating system</td>
<td>operating system</td>
<td>operating system</td>
</tr>
<tr>
<td>Backed by</td>
<td>cache</td>
<td>main memory</td>
<td>disk</td>
<td>disk</td>
<td>disk or tape</td>
</tr>
</tbody>
</table>

Movement between levels of storage hierarchy can be explicit or implicit

- **Cache managed by hardware. Makes main memory appear much faster.**
- **Disks are several orders of magnitude slower.**
Memory Management

- To execute a program all (or part) of the instructions must be in memory.
- All (or part) of the data that is needed by the program must be in memory.
- Memory management determines what is in memory and when:
  - Optimizing CPU utilization and computer response to users.
- Memory management activities:
  - Keeping track of which parts of memory are currently being used and by whom.
  - Deciding which processes (or parts thereof) and data to move into and out of memory.
  - Allocating and deallocating memory space as needed.
Storage Management

• OS provides uniform, logical view of information storage
  – Abstracts physical properties to logical storage unit - file
  – Each medium is controlled by device (i.e., disk drive, tape drive)
    • Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)

• File-System management
  – Files usually organized into directories
  – Access control on most systems to determine who can access what
  – OS activities include
    • Creating and deleting files and directories
    • Primitives to manipulate files and directories
    • Mapping files onto secondary storage
    • Backup files onto stable (non-volatile) storage media
Mass-Storage Management

• Usually disks used to store data that does not fit in main memory or data that must be kept for a “long” period of time
• Entire speed of computer operation hinges on disk subsystem and its algorithms
• OS activities
  – Free-space management
  – Storage allocation
  – Disk scheduling
• Some storage need not be fast
  – Tertiary storage includes optical storage, magnetic tape
  – Still must be managed – by OS or applications
  – Varies between WORM (write-once, read-many-times) and RW (read-write)
Migration of data “A” from Disk to Register

- Multitasking environments must be careful to use most recent value, no matter where it is stored in the storage hierarchy

- Multiprocessor environment must provide cache coherency in hardware such that all CPUs have the most recent value in their cache
- Distributed environment situation even more complex
  - Several copies of a datum can exist
  - Various solutions covered in Chapter 19 (*will not get to it*)
CS370 Operating Systems

Colorado State University
Yashwant K Malaiya
Spring 2019 OS Structures

Slides based on
• Text by Silberschatz, Galvin, Gagne
• Various sources
Chap2: Operating-System Structures

Objectives:

• Services OS provides to users, processes, and other systems
• Structuring an operating system
• How operating systems are designed and customized and how they boot
OS Services for the User 1/3

• Operating systems provide an environment for execution of programs and services to programs and users
  – **User interface** - Almost all operating systems have a user interface (UI).
    • Varies between *Command-Line (CLI)*, *Graphics User Interface (GUI)*, *Batch*
  – **Program execution** - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
  – **I/O operations** - A running program may require I/O, which may involve a file or an I/O device
– **File-system operations** - read and write files and directories, create and delete them, search them, list file information, permission management.

– **Communications** – Processes may exchange information, on the same computer or between computers over a network
  
  • via shared memory or through message passing (packets moved by the OS)

– **Error detection** – OS needs to be constantly aware of possible errors
  
  • May occur in the CPU and memory hardware, in I/O devices, in user program
  
  • For each type of error, OS should take the appropriate action to ensure correct and consistent computing
OS services for system 3/3 (Cont.)

- OS functions for ensuring the efficient resource sharing
  - **Resource allocation** - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
    - Many types of resources - CPU cycles, main memory, file storage, I/O devices.
  - **Accounting** - To keep track of which users use how much and what kinds of computer resources
  - **Protection and security** - concurrent processes should not interfere with each other
    - **Protection** involves ensuring that all access to system resources is controlled
    - **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts
A View of Operating System Services

user and other system programs

<table>
<thead>
<tr>
<th>GUI</th>
<th>batch</th>
<th>command line</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

user interfaces

system calls

- program execution
- I/O operations
- file systems
- communication
- resource allocation
- accounting
- error detection
- protection and security

services

operating system

hardware
CLI or command interpreter allows direct command entry

- Sometimes implemented in kernel, sometimes by systems program
- Sometimes multiple flavors implemented – shells
- Primarily fetches a command from user and executes it
- Sometimes commands built-in, sometimes just names of programs
  - If the latter, adding new features doesn’t require shell modification

Ex:
Windows: command prompt
Linux: bash
Shell Command Interpreter

A bash session

```
Last login: Sat Aug 27 22:09:08 on ttys000
Ys-MacBook-Air:~ ymalaiya$ echo $0
-bash
Ys-MacBook-Air:~ ymalaiya$ pwd
/Users/ymalaiya
Ys-MacBook-Air:~ ymalaiya$ ls
270  Desktop  Downloads  Music  android-SDKs
Applications  Dialcom  Library  Pictures
DLID Books  Documents  Movies  Public
Ys-MacBook-Air:~ ymalaiya$ w
22:14 up 1:12, 2 users, load averages: 1.15 1.25 1.27
USER   TTY   FROM     LOGIN@  IDLE  WHAT
ymalaiya console   -       21:02  1:11 -
ymalaiya s000      -       22:14  - w
Ys-MacBook-Air:~ ymalaiya$ ps
   PID  TTY      TIME CMD
       594 ttys000 0:00.02 -bash
Ys-MacBook-Air:~ ymalaiya$ iostat 5
                   disk0    cpu load average
                 KB/t  tps  MB/s us  sy  id  1m  5m  15m
               36.76 17 0.60  5  3  92  1.42 1.31 1.28
^C
Ys-MacBook-Air:~ ymalaiya$ ping colostate.edu
PING colostate.edu (129.82.103.93): 56 data bytes
64 bytes from 129.82.103.93: icmp_seq=0 ttl=116 time=46.069 ms
64 bytes from 129.82.103.93: icmp_seq=1 ttl=116 time=41.327 ms
64 bytes from 129.82.103.93: icmp_seq=2 ttl=116 time=58.673 ms
64 bytes from 129.82.103.93: icmp_seq=3 ttl=116 time=44.750 ms
64 bytes from 129.82.103.93: icmp_seq=4 ttl=116 time=48.336 ms
^C
--- colostate.edu ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 41.327/47.831/58.673/5.877 ms
Ys-MacBook-Air:~ ymalaiya$  
```
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pwd</code></td>
<td>print Working directory</td>
</tr>
<tr>
<td><code>ls -l</code></td>
<td>Files in the working dir –long format</td>
</tr>
<tr>
<td><code>cd dirpath</code></td>
<td>Change to dirpath dir</td>
</tr>
<tr>
<td><code>..</code></td>
<td><code>~username</code>/ This dir, <code>upper</code>, usename’s home, <code>root</code></td>
</tr>
<tr>
<td><code>cp f1 d1</code></td>
<td>Copy f1 to dir d1</td>
</tr>
<tr>
<td><code>mv f1 d1</code></td>
<td>Move f1 to d1</td>
</tr>
<tr>
<td><code>rm f1 f2</code></td>
<td>Remove f1, f2</td>
</tr>
<tr>
<td><code>mkdir d1</code></td>
<td>Create directory d1</td>
</tr>
<tr>
<td><code>which x1</code></td>
<td>Path for executable file x1</td>
</tr>
<tr>
<td><code>man cm</code></td>
<td>Manual entry or help with command cm</td>
</tr>
<tr>
<td><code>ls &gt; f.txt</code></td>
<td>Redirect command std output to f.txt, <code>&gt;&gt;</code> to append</td>
</tr>
<tr>
<td><code>sort &lt; list.txt</code></td>
<td>Std input from file</td>
</tr>
<tr>
<td>`ls -l</td>
<td>less`</td>
</tr>
</tbody>
</table>
## Common bash commands 2/2

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>echo $((expression))</code></td>
<td>Evaluate expression</td>
</tr>
<tr>
<td><code>echo $PATH</code></td>
<td>Show PATH</td>
</tr>
<tr>
<td><code>echo $SHELL</code></td>
<td>Show default shell</td>
</tr>
<tr>
<td><code>chmod 755 dir</code></td>
<td>Change dir permissions to 755</td>
</tr>
<tr>
<td><code>jobs</code> <code>ps</code></td>
<td>List jobs for current shell, processes in the system</td>
</tr>
<tr>
<td><code>kill id</code></td>
<td>Kill job or process with given id</td>
</tr>
<tr>
<td><code>cmd &amp;</code></td>
<td>Start job in background</td>
</tr>
<tr>
<td><code>fg id</code></td>
<td>Bring job id to foreground</td>
</tr>
<tr>
<td><code>ctrl-z</code> followed by <code>bg</code> or <code>fg</code></td>
<td>Suspend job and put it in background</td>
</tr>
<tr>
<td><code>w</code> <code>who</code></td>
<td>Who is logged on</td>
</tr>
<tr>
<td><code>ping ipaddr</code></td>
<td>Get a ping from ipadd</td>
</tr>
<tr>
<td><code>ssh user@host</code></td>
<td>Connect to host as user</td>
</tr>
<tr>
<td><code>grep pattern files</code></td>
<td>Search for pattern in files</td>
</tr>
<tr>
<td><code>Ctrl-c</code></td>
<td>Halt current command</td>
</tr>
</tbody>
</table>
User-friendly **desktop** metaphor interface

- Usually mouse, keyboard, and monitor
- **Icons** represent files, programs, actions, etc
- Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a **folder**))
- Invented at Xerox PARC in 1973

**Most systems now include both CLI and GUI interfaces**

- Microsoft Windows is GUI with CLI “command” shell
- Apple Mac OS X is “Aqua” GUI interface with UNIX kernel underneath and shells available
- Unix and Linux have CLI with optional GUI interfaces (CDE, KDE, GNOME)
Touchscreen Interfaces

- Touchscreen devices require new interfaces
  - Mouse not possible or not desired
  - Actions and selection based on gestures
  - Virtual keyboard for text entry
  - Voice commands.
The Mac OS X GUI
System Calls

• Programming interface to the services provided by the OS
• Typically written in a high-level language (C or C++)
• Mostly accessed by programs via a high-level Application Programming Interface (API) rather than direct system call use
• Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)

Note that the system-call names used throughout our text are generic.
Example of System Calls

- System call sequence to copy the contents of one file to another file

Example System Call Sequence
- Acquire input file name
- Write prompt to screen
- Accept input
- Acquire output file name
- Write prompt to screen
- Accept input
- Open the input file
- if file doesn't exist, abort
- Create output file
- if file exists, abort
- Loop
  - Read from input file
  - Write to output file
  - Until read fails
- Close output file
- Write completion message to screen
- Terminate normally
Example of Standard API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```
on the command line. A description of this API appears below:

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count)

return function parameters

value name
```

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `int fd` — the file descriptor to be read
- `void *buf` — a buffer where the data will be read into
- `size_t count` — the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns -1.

unistd.h header file provides access to the POSIX API
The caller need know nothing about how the system call is implemented

- Just needs to obey API and understand what OS will do as a result call
- Most details of OS interface hidden from programmer by API
  - Managed by run-time support library (set of functions built into libraries included with compiler)

System call implementation examples:

- LC-3 Trap x21 (OUT) code in Patt & Patel (see slide 22)
- Identified by a number that leads to address of the routine
- Arguments provided in designated registers
- Linux x86_64 table, code snippets
API – System Call – OS Relationship

user application

open ()

user mode

system call interface

kernel mode

Trap vector table in LC3

Implementation of open ()

return
Examples of Windows and Unix System Calls

<table>
<thead>
<tr>
<th>Windows</th>
<th>Unix</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Process Control</strong></td>
<td></td>
</tr>
<tr>
<td>CreateProcess()</td>
<td>fork()</td>
</tr>
<tr>
<td>ExitProcess()</td>
<td>exit()</td>
</tr>
<tr>
<td>WaitForSingleObject()</td>
<td>wait()</td>
</tr>
<tr>
<td><strong>File Manipulation</strong></td>
<td></td>
</tr>
<tr>
<td>CreateFile()</td>
<td>open()</td>
</tr>
<tr>
<td>ReadFile()</td>
<td>read()</td>
</tr>
<tr>
<td>WriteFile()</td>
<td>write()</td>
</tr>
<tr>
<td>CloseHandle()</td>
<td>close()</td>
</tr>
<tr>
<td><strong>Device Manipulation</strong></td>
<td></td>
</tr>
<tr>
<td>SetConsoleMode()</td>
<td>ioctl()</td>
</tr>
<tr>
<td>ReadConsole()</td>
<td>read()</td>
</tr>
<tr>
<td>WriteConsole()</td>
<td>write()</td>
</tr>
<tr>
<td><strong>Information Maintenance</strong></td>
<td></td>
</tr>
<tr>
<td>GetCurrentProcessID()</td>
<td>getpid()</td>
</tr>
<tr>
<td>SetTimer()</td>
<td>alarm()</td>
</tr>
<tr>
<td>Sleep()</td>
<td>sleep()</td>
</tr>
<tr>
<td><strong>Communication</strong></td>
<td></td>
</tr>
<tr>
<td>CreatePipe()</td>
<td>pipe()</td>
</tr>
<tr>
<td>CreateFileMapping()</td>
<td>shmat()</td>
</tr>
<tr>
<td>MapViewOfFile()</td>
<td>mmap()</td>
</tr>
<tr>
<td><strong>Protection</strong></td>
<td></td>
</tr>
<tr>
<td>SetFileSecurity()</td>
<td>chmod()</td>
</tr>
<tr>
<td>InitializeSecurityDescriptor()</td>
<td>umask()</td>
</tr>
<tr>
<td>SetSecurityDescriptorGroup()</td>
<td>chown()</td>
</tr>
</tbody>
</table>
Standard C Library Example

• C program invoking `printf()` library call, which calls `write()` system call
Example OS: MS-DOS ’81..

- Single-tasking
- Shell invoked when system booted
  - No process created
- Single memory space
- Loads program into memory, overwriting all but the kernel
- Program exit -> shell reloaded

At system startup
running a program
Example: xBSD ‘93 Berkely

- Unix ‘73 variant, inherited by several later OSs
- Multitasking
- User login -> invoke user’s choice of shell
- Shell executes fork() system call to create process
  - Executes exec() to load program into process
  - Shell waits for process to terminate or continues with user commands
- Process exits with:
  - code = 0 – no error
  - code > 0 – error code
POSIX

- POSIX: Portable Operating Systems Interface for UNIX  
  Pronounced *pahz-icks*

- **POSIX.1** published in 1988

- Final POSIX standard: Joint document
  - Approved by IEEE & Open Group End of 2001
  - ISO/IEC approved it in November 2002
  - Most recent *IEEE Std 1003.1-2008, 2016 Edition*

- Most OSs are *mostly POSIX-compliant*