CS370 Operating Systems
Colorado State University
Yashwant K Malaiya
Spring 21 Lecture 4
OS Structures/Processes

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

- A chip can have one or more processors (CPU, core) and possibly more components.

- Where is the DMA controller? Separate chip or same chip?

- **Kernel vs OS:** Kernel: process/memory/file/IO management, OS can include UI, libraries etc.

- **Why User vs kernel mode?** Because users can’t be trusted.

- Where are registers, Cache and main memory, physically?

- **Trap routines, interrupts**

- Good that you are thinking about these.
  - Threads vs processes, Scheduling, Memory management: we will study in detail
  - Multi-level caches: how are they implemented?

Note: TA office hours are available. Help session Slides: Schedule, Video: Teams
K-scale: Amount of information/storage

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

- **Milliseconds**, **microseconds**, **nanoseconds**, **pico**seconds $10^{-3}, 10^{-6}, 10^{-9}, 10^{-12}$
CS370 Operating Systems
Colorado State University
Yashwant K Malaiya
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
• 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 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
Viewing Processes

MAC: look at processes

**Activity Monitor** User Guide > CPU, **Process**, threads, PID etc. info about a process

<table>
<thead>
<tr>
<th>Process Name</th>
<th>% CPU</th>
<th>CPU Time</th>
<th>Threads</th>
<th>Idle Wake Ups</th>
<th>% CPU</th>
<th>CPU Time</th>
<th>PID</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>WindowServer</td>
<td>22.0</td>
<td>2:23:52.20</td>
<td>14</td>
<td>59</td>
<td>5.6</td>
<td>7:46.00</td>
<td>144</td>
<td>_windowserver</td>
</tr>
<tr>
<td>Activity Monitor</td>
<td>10.8</td>
<td>19:22:70</td>
<td>5</td>
<td>2</td>
<td>0.0</td>
<td>0.00</td>
<td>3782</td>
<td>julietstine</td>
</tr>
<tr>
<td>lsweb-agent</td>
<td>6.6</td>
<td>5:53:21</td>
<td>13</td>
<td>0</td>
<td>0.0</td>
<td>0.00</td>
<td>3196</td>
<td>julietstine</td>
</tr>
<tr>
<td>kernel_task</td>
<td>6.3</td>
<td>33:22:18</td>
<td>224</td>
<td>338</td>
<td>0.5</td>
<td>0.00</td>
<td>9</td>
<td>root</td>
</tr>
<tr>
<td>Messages</td>
<td>4.5</td>
<td>21:53:09</td>
<td>4</td>
<td>53</td>
<td>0.0</td>
<td>0.00</td>
<td>5534</td>
<td>julietstine</td>
</tr>
<tr>
<td>systemd</td>
<td>3.1</td>
<td>18:20:42</td>
<td>3</td>
<td>0</td>
<td>0.0</td>
<td>0.00</td>
<td>366</td>
<td>root</td>
</tr>
<tr>
<td>ScreenSharingAgent</td>
<td>3.0</td>
<td>2:06:50</td>
<td>6</td>
<td>7</td>
<td>18.4</td>
<td>19.49</td>
<td>2426</td>
<td>julietstine</td>
</tr>
<tr>
<td>macterminalhelper</td>
<td>1.9</td>
<td>1:28:40</td>
<td>6</td>
<td>0</td>
<td>0.0</td>
<td>0.00</td>
<td>3353</td>
<td>julietstine</td>
</tr>
<tr>
<td>cerebrightness</td>
<td>1.2</td>
<td>43.91</td>
<td>6</td>
<td>23</td>
<td>0.0</td>
<td>0.00</td>
<td>138</td>
<td>root</td>
</tr>
<tr>
<td>launchd-nosave</td>
<td>1.0</td>
<td>1:08:11</td>
<td>6</td>
<td>0</td>
<td>0.0</td>
<td>0.00</td>
<td>114</td>
<td>root</td>
</tr>
<tr>
<td>launchd-tkachix</td>
<td>0.7</td>
<td>6.75</td>
<td>5</td>
<td>4</td>
<td>0.0</td>
<td>0.00</td>
<td>7568</td>
<td>julietstine</td>
</tr>
<tr>
<td>tccd</td>
<td>0.7</td>
<td>35.64</td>
<td>3</td>
<td>0</td>
<td>0.0</td>
<td>0.00</td>
<td>151</td>
<td>root</td>
</tr>
<tr>
<td>launchd-development</td>
<td>0.6</td>
<td>2:21:40</td>
<td>4</td>
<td>0</td>
<td>0.0</td>
<td>0.00</td>
<td>1</td>
<td>root</td>
</tr>
<tr>
<td>screen-sharing</td>
<td>0.6</td>
<td>47.09</td>
<td>7</td>
<td>0</td>
<td>0.0</td>
<td>0.00</td>
<td>7425</td>
<td>root</td>
</tr>
<tr>
<td>SSMenuAgent</td>
<td>0.6</td>
<td>1:03:42</td>
<td>5</td>
<td>3</td>
<td>0.0</td>
<td>0.00</td>
<td>4272</td>
<td>julietstine</td>
</tr>
<tr>
<td>loginwindow</td>
<td>0.5</td>
<td>53.89</td>
<td>4</td>
<td>0</td>
<td>0.0</td>
<td>0.00</td>
<td>153</td>
<td>julietstine</td>
</tr>
<tr>
<td>powermetrics</td>
<td>0.4</td>
<td>23.76</td>
<td>1</td>
<td>0</td>
<td>0.0</td>
<td>0.00</td>
<td>2250</td>
<td>root</td>
</tr>
<tr>
<td>trustid</td>
<td>0.4</td>
<td>1:20.22</td>
<td>2</td>
<td>0</td>
<td>0.0</td>
<td>0.00</td>
<td>174</td>
<td>root</td>
</tr>
</tbody>
</table>

Click to quit a process.
Click a column heading to sort the list.
Search for a process.

Windows: Open **Task Manager**.

See information about the number of open processes and threads.
A View of Operating System Services

Diagram:

- User and other system programs
  - GUI
  - Batch
  - Command line
  - User interfaces

- System calls
  - Program execution
  - I/O operations
  - File systems
  - Communication
  - Resource allocation
  - Accounting
  - Error detection
  - Protection and security

- 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
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
0  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 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>This dir, upper, username’s home, root</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    help 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, &gt;&gt; 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>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 who</code></td>
<td>Who is logged on</td>
</tr>
<tr>
<td><code>ping ipadd</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> (shows as <code>^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:
1. Acquire input file name
2. Write prompt to screen
3. Accept input
4. Acquire output file name
5. Write prompt to screen
6. Accept input
7. Open the input file
   - If file doesn't exist, abort
8. Create output file
   - If file exists, abort
9. Loop
   - Read from input file
   - Write to output file
   - Until read fails
10. Close output file
11. Write completion message to screen
12. 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)
```

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.
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

Implementation of open ()
system call

Trap vector table in LC3

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>shmget()</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
POSIX

• POSIX: Portable Operating Systems Interface for UNIX for system commands 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 1003.1-2017 Edition

• Most OSs are mostly POSIX-compliant

• We will use a few POSIX-compliant system commands
Example OS: MS-DOS ‘81..

- Single-tasking
- Shell invoked when system booted
- Simple method to run program
  - 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
System Programs 1/4

• System programs provide a convenient environment for program development and execution. They can be divided into:
  – File manipulation
  – Status information sometimes stored in a File modification
  – Programming language support
  – Program loading and execution
  – Communications
  – Background services
  – Application programs

• Most users’ view of the operation system is defined by system programs, not the actual system calls
• Provide a convenient environment for program development and execution
  – Some of them are simply user interfaces to system calls; others are considerably more complex

• **File management** - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories

• **Status information**
  – Some ask the system for info - date, time, amount of available memory, disk space, number of users
  – Others provide detailed performance, logging, and debugging information
  – Typically, these programs format and print the output to the terminal or other output devices
  – Some systems implement a **registry** - used to store and retrieve configuration information
System Programs 3/4

• **File modification**
  – Text editors to create and modify files
  – Special commands to search contents of files or perform transformations of the text

• **Programming-language support** - Compilers, assemblers, debuggers and interpreters sometimes provided

• **Program loading and execution** - Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language

• **Communications** - Provide the mechanism for creating virtual connections among processes, users, and computer systems
  – Allow users to send messages to one another’s screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another
• **Background Services**
  - Launch at boot time
    - Some for system startup, then terminate
    - Some from system boot to shutdown
  - Provide facilities like disk checking, process scheduling, error logging, printing
  - Run in user context not kernel context
  - Known as services, subsystems, daemons

• **Application programs**
  - Don’t pertain to system
  - Run by users
  - Not typically considered part of OS
  - Launched by command line, mouse click, finger poke
• General-purpose OS is very large program
• Various ways to structure ones
  – Simple structure – MS-DOS. not modular
  – More complex – UNIX.
    • Kernel+systems programs
  – Layered – an abstraction
  – Microkernel –Mach: kernel is minimal
  – hybrid

"LINUX is obsolete".
CS370 OS  Ch3  Processes

• Process Concept: a program in execution
• Process Scheduling
• Processes creation and termination
• Interprocess Communication using shared memory and message passing
• An operating system executes a variety of programs:
  – Batch system – jobs
  – Time-shared systems – user programs or tasks
• Textbook uses the terms job and process almost interchangeably
• Process – a program in execution; process execution must progress in sequential fashion. Includes
  – The program code, also called “text section”
  – Current activity including program counter, processor registers
  – Stack containing temporary data
    • Function parameters, return addresses, local variables
  – Data section containing global variables
  – Heap containing memory dynamically allocated during run time
• Program is **passive** entity stored on disk (**executable file**), process is **active**
  – Program becomes process when executable file loaded into memory
• Execution of program started via GUI mouse clicks, command line entry of its name, etc
• One program can be several processes
  – Consider multiple users executing the same program
Process in Memory
Process State

• As a process executes, it changes state
  – new: The process is being created
  – running: Instructions are being executed
  – waiting: The process is waiting for some event to occur
  – ready: The process is waiting to be assigned to a processor
  – terminated: The process has finished execution
Meanwhile, on an ordinary Linux kernel...

What's going on with these zombie processes?

Their parent is too busy to get any notifications...

Daniel Stori {turnoff.us}
Diagram of Process State

Transitions:
- **Ready to Running**: scheduled by scheduler
- **Running to Ready**: scheduler picks another process, back in ready queue
- **Running to Waiting** (Blocked): process blocks for input/output
- **Waiting to Ready**: I/O or event done
Information associated with each process (also called **task control block**)

- Process state – running, waiting, etc
- Program counter – location of instruction to next execute
- CPU registers – contents of all process-centric registers
- CPU scheduling information – priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files
CPU Switch From Process to Process

- **Process $P_0$**
  - Executing
  - If interrupted or system call, save state into PCB$_0$
    - Continue...

- **Operating System**
  - Idle

- **Process $P_1$**
  - Executing
  - If interrupted or system call, save state into PCB$_1$
    - Continue...
  - Reload state from PCB$_0$
  - Idle