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.

• 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, picoseconds $10^{-3}, 10^{-6}, 10^{-9}, 10^{-12}$
CS370 Operating Systems
Colorado State University
Yashwant K Malaiya
OS Structures

Slides based on
- Text by Silberschatz, Galvin, Gagne
- Various sources
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

- User interface -
  - Varies between Command-Line (CLI), Graphics User Interface (GUI), Batch
- Program execution
- I/O operations
- File-system operations
- Communications
- Error detection
- Resource allocation
- Accounting
- Protection and security

MAC: look at processes

Activity Monitor > CPU

Process, threads, PID etc.
info about a process
## A View of Operating System Services

<table>
<thead>
<tr>
<th>user and other system programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUI</td>
</tr>
<tr>
<td>user interfaces</td>
</tr>
</tbody>
</table>

### system calls

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

### 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
### Common bash commands 1/2

<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</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>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<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 bg or fg</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 ipaddr</code></td>
<td>Get a ping from ipaddr</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>Ctrl-c (shows as ^C)</td>
<td>Halt current command</td>
</tr>
</tbody>
</table>
User-operating System Interface - GUI

• 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.
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

Implementation of open()

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
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

![Diagram](image)

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: 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*
• 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
Operating System Design

• 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

Tanenbaum–Torvalds debate: (January 29, 1992). "LINUX is obsolete".
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
Process Concept (Cont.)

• 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

- stack
- heap
- data
- text
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
Process Control Block (PCB)

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 $P_0$  
operating system  
process $P_1$

executing  
interrupt or system call
save state into PCB$_0$  
  
  

reload state from PCB$_1$

idle
executing

idle

executing

interrupt or system call
save state into PCB$_1$  
  
  

reload state from PCB$_0$
Demonstration: Processes

- **Mac**: apps > utilities > activity monitor > CPU etc.
    - See information about processes
    - Name, PID, threads, details ..

- **Windows 10** Ctrl+Alt+Del