CS370 Operating Systems
Colorado State University
Yashwant K Malaiya
Fall 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?

• Scheduling, Memory management, storage management? Good that you are thinking about these.

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 $10^{-3}$, $10^{-6}$, $10^{-9}$, $10^{-12}$
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

  ![Diagram](image)

  - hard disk → main memory → cache → hardware register

- 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
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 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
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 -
ys-MacBook-Air:$ ymalaiya$ w
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$
```
### 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>. .. ~username /</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>jobs 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 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>Ctrl-c</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 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)
```

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.
System Call Implementation

- 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

open ()

Implementation of open ()

system call

return

Trap vector table in LC3
## Examples of Windows and Unix System Calls

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

Colorado State University
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*
CS370 Operating Systems
Colorado State University
Yashwant K Malaiya
Back from ICQ
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
• **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

Tanenbaum–Torvalds debate: (January 29, 1992). "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
Process Concept

• 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 runtime
• 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

max

stack

heap

data

text