CSE325 Principles of Operating Systems

Operating System Services

David Duggan
dduggan@sandia.gov

New Mexico Tech

January 22, 2013
Reading Assignment 3

- Chapter 3, due 01/29
What Categories of Services Might be Provided?

- Program execution
- Protection
- Process management
- Memory management
- Storage management
- Communication
- Security
OSes: Program Execution

- **Multiprogramming** needed for efficiency of utilization
  - 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 the 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
OS Operations

- **Interrupt** driven by hardware
- Software error or request creates *exception* or *trap*
  - Division by zero, request for operating system service
- Other process problems include infinite loop, 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
Transition from User to Kernel Mode

- Timer to prevent infinite loop / process hogging resources
  - Set interrupt after specific period
  - Operating system decrements counter
  - When counter reaches zero, generate an interrupt
  - Set up before scheduling process to regain control or terminate program that exceeds allotted time
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 reclaiming of any reusable resources

- Single-threaded process has one program counter specifying the location of the 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
Process Management Activities

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 communication
- Providing mechanisms for process synchronization
- Providing mechanisms for deadlock handling
Memory Management

- 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

- All data in memory before and after processing
- All instructions in memory to allow execution
- Memory management determines what is in memory
  - Optimizing CPU utilization & computer response to users
Storage Management

- **File-System management**
  - Files usually organized in 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

- **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)
I/O Subsystem

- I/O subsystem responsible for
  - Memory management of I/O including
    - **Buffering**: storing data temporarily while it is being transferred
    - **Caching**: storing parts of data in faster storage for performance
    - **Spooling**: the overlapping of output of one job with input of other jobs
  - General device-driver interface
  - Drivers for specific hardware devices

- One purpose of OS is to hide peculiarities of hardware devices from the user
Protection and Security

- **Protection** – any mechanism for controlling access of processes or users to resources defined by the OS

- **Security** – defense of the system against internal and external attacks
  - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service

- Systems generally first distinguish among users, to determine who can do what
  - User identities (user IDs, security IDs) include name and associated number, one per user
  - User ID then associated with all files, processes of that user to determine access control
  - Group identifier (group ID) allows set of users to be defined and controls managed, then also associated with each process, file
  - **Privilege escalation** allows user to change to an effective ID with more rights
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>
<tr>
<td>user interfaces</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

system calls

<table>
<thead>
<tr>
<th>program execution</th>
<th>I/O operations</th>
<th>file systems</th>
<th>communication</th>
<th>resource allocation</th>
<th>accounting</th>
<th>protection and security</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>error detection</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

services

operating system

hardware
Operating System Services

Some services provide functions helpful to the user:

- **User interface** - Almost all operating systems have a user interface (UI)
  - 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 manipulation** - The file system is of particular interest. Obviously, programs need to read and write files and directories, create and delete them, change them, search them, list file information, perform permission management.
Operating System Services (Cont.)

- **Communications** – Processes may exchange information, on the same computer or between computers over a network
  - Communications may be 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
  - Debugging facilities can greatly enhance the user’s and programmer’s abilities to efficiently use the system
For efficient operation of system via resource sharing

- **Resource allocation** - When multiple users / jobs running concurrently, resources must be allocated to each of them

- **Accounting** - To keep track of which users use, how much, and what kinds of computer resources

- **Protection and security**
  - **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

- If a system is to be protected and secure, precautions must be instituted throughout it. A chain is only as strong as its weakest link.
Bourne Shell Command Interpreter

```
-$(/var/tmp/system-contents/scripts)# swap -sh

  total: 1.1G allocated + 190M reserved = 1.3G used, 1.6G available

-$(/var/tmp/system-contents/scripts)# uptime

  12:53am up 9 min(s), 3 users, Load average: 33.29, 67.68, 36.81

-$(/var/tmp/system-contents/scripts)# w

  4:07pm up 17 day(s), 15:24, 3 users, Load average: 0.09, 0.11, 8.66

User    tty     login@ idle  JCPU  PCPU  what
root    console 15Jun0718days 1  /usr/bin/ssh-agent -- /usr/bin/

n/d
root    pts/3   15Jun07    18    4 w
root    pts/4   15Jun0718days w
```

User Operating System Interface - CLI

Command-Line Interface (CLI) allows direct command entry

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

- Many systems now include both CLI and GUI interfaces
  - Microsoft Windows is GUI; CLI is “command” shell
  - Apple Mac OS X has “Aqua” GUI; UNIX kernel underneath and multiple shells available
  - Solaris has multiple GUIs; CLI is multiple shells
System Calls

- Programming interface to the services provided by OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level Application Program 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)
- Why use APIs rather than system calls?
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

**ReadFile()**: Win32 API—a function for reading from a file

```c
BOOL ReadFile (HANDLE file, LPVOID buffer, DWORD bytesToRead, LPDWORD bytesRead, LPOVERLAPPED ovl);
```

- A description of the parameters passed to ReadFile()
  - HANDLE file—the internal handle of the file to be read
  - LPVOID buffer—a buffer where the data will be read into and written from
  - DWORD bytesToRead—the number of bytes to be read into the buffer
  - LPDWORD bytesRead—the number of bytes read during the last read
  - LPOVERLAPPED ovl—indicates if overlapped I/O is being used
API – System Call – OS Relationship

user application

open ()

system call interface

user mode

kernel mode

i

Implementation of open ()

system call

return
System Call Implementation

- Typically, a number associated with each system call
  - System-call interface maintains a table indexed according to these numbers

- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values

- 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 of the 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)
Standard C Library Example

- C program invoking `printf()` library call, which calls `write()` system call
System Call Parameter Passing

- Often, more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and call

- Methods used to pass parameters to OS
  - Simplest: pass the parameters in registers
    - In some cases, may be more parameters than registers
  - Parameters stored in a block, or table, in memory, and address of block passed as a parameter in a register
    - This approach taken by Linux and Solaris
  - Parameters placed, or pushed, onto the stack by the program and popped off the stack by the operating system
  - Block and stack methods do not limit the number or length of parameters being passed
Parameter Passing via Table

X: parameters for call
load address X
system call 13
user program

register

use parameters from table X

code for system call 13

operating system
Types of System Calls

- Process control
- File management
- Device management
- Information maintenance
- Communications
System Programs

- System programs provide a convenient environment for program development and execution. Including:
  - File manipulation
  - Status information
  - Programming language support
  - Program loading and execution
  - Communications
  - Application programs

- Most users’ view of the operating system is defined by system programs, not the actual system calls