Today's Plan

Upcoming:

- Assignment 1

Last time:

- Operating system duties

Today's topics:

- File/Secondary Storage management
- I/O System
- Protection/security
- Computing environments
- Operating System Services
- User Operating System Interface
- System Calls

System Components – File Management

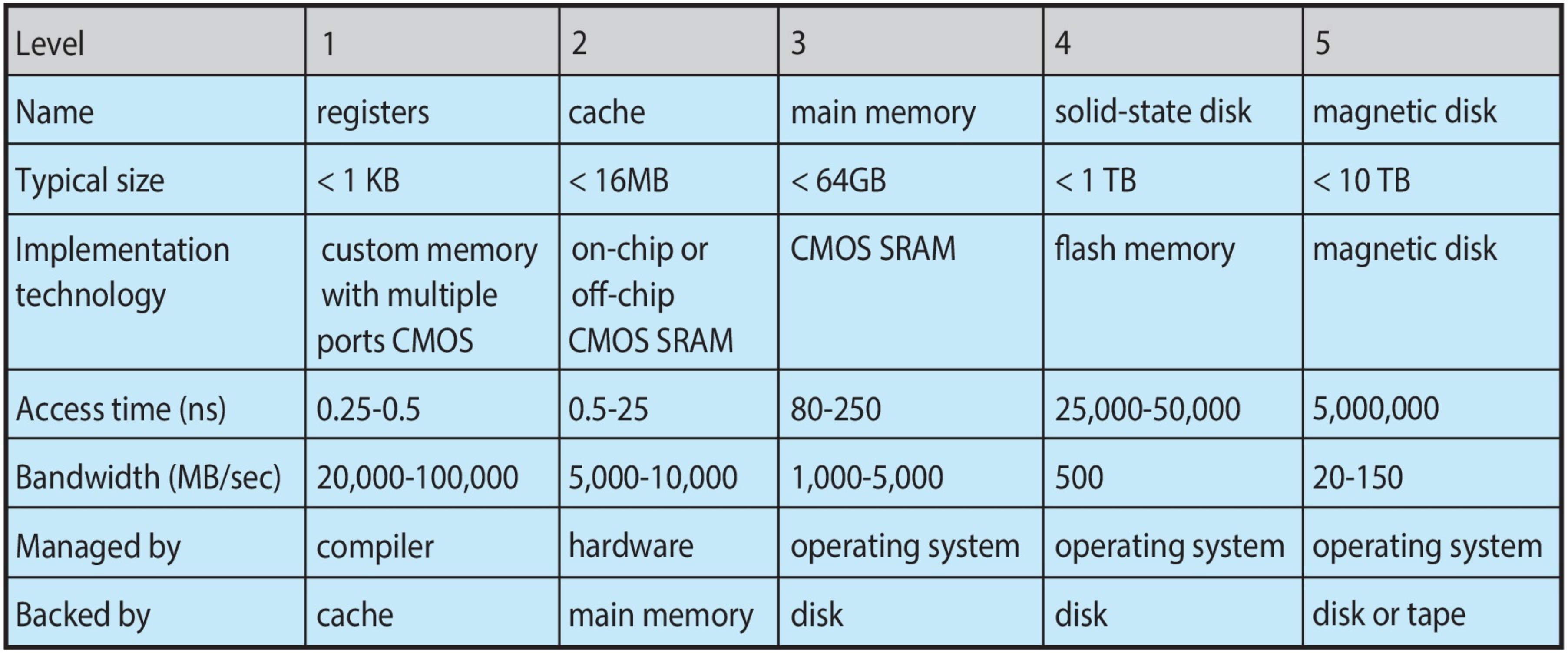
- A file is a collection of related information defined by its creator.
- The operating system is responsible for the following activities in connection with file management:
  - File/directory creation and deletion
  - Support of primitives for manipulating files and directories
  - Access control available on most systems
  - Mapping files onto secondary storage
  - File backup on stable (non-volatile) storage media

System Components – Secondary Storage Management

- Since main memory (*primary storage*) is volatile and too small to accommodate all data and programs permanently, the computer system must provide *secondary storage* to back up main memory.
- Most modern computer systems use drives as the principle storage medium, for both programs and data.
- The operating system is responsible for the following activities in connection with disk management:
  - Free space management
  - Storage allocation
  - Disk scheduling

Performance of Various Levels of Storage

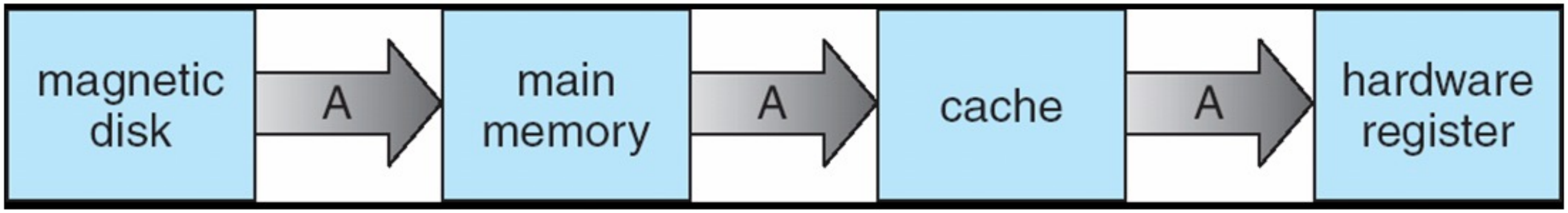
- Movement between levels of storage hierarchy can be explicit or implicit



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

Migration of Integer 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

System Components – I/O System Management

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

The I/O system consists of:

- A buffer-caching system
- A general device driver interface
  - Device driver: a set of interrupt handler/subroutines for a device controller
- Drivers for specific hardware devices

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
  - E.g. denial-of-service attacks, worms, viruses, identity theft, theft of service, etc.

Protection and Security

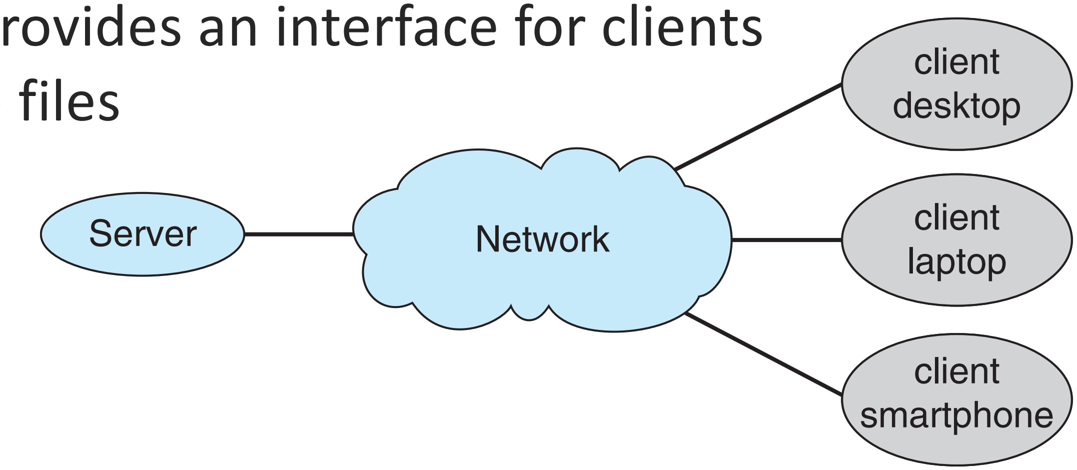
- Systems generally first distinguish among users, to determine who can do what
  - User identities (**user IDs**, security IDs) include name and associated number, one per user
  - User ID then associated with all files, processes of that user to determine access control
  - Group identifier (**group 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 effective ID with more rights

Computing Environments – Traditional & Mobile

- Traditional
  - Stand-alone general-purpose computers
  - Range from network computers (thin clients) to powerful laptops/desktops
- Mobile
  - Handheld smartphones, tablets, etc.
  - OS must support many features enabled by sensors (GPS, gyroscope, cameras)
  - Allows for new types of apps like augmented reality

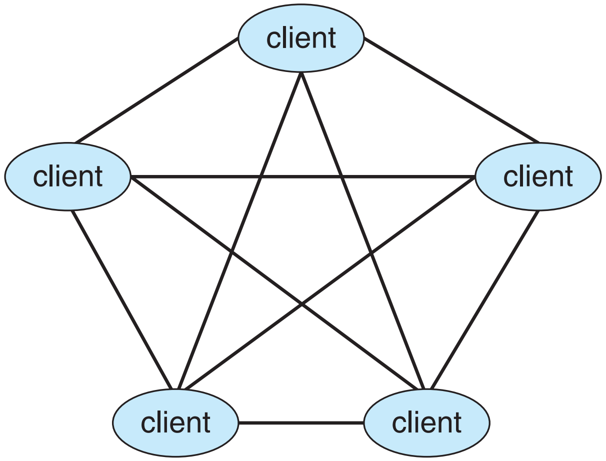
Computing Environments – Client Server

- Client-Server Computing
  - Dumb terminals supplanted by smart PCs
  - Many systems now **servers**, responding to requests generated by **clients**
    - **Compute-server system** provides an interface to client to request services (e.g. database)
    - **File-server system** provides an interface for clients to store and retrieve files



Peer-to-Peer Computing

- Another model for a distributed system
- P2P does not distinguish clients and servers
  - All nodes are considered peers
  - May each act as client, server, or both
  - Node must join P2P network
    - Registers its service with central lookup service on network, or
    - Broadcast request for service and respond to requests for service via **discovery protocol**
  - Examples include *Napster, Gnutella*, and VOIP services



Computing Environments – Cloud Computing

- Delivers computing, storage, and apps as a service across a network
  - E.g. Amazon EC2 has thousands of servers, millions of virtual machines, petabytes of storage available via the internet – pay based on usage
  - **Public** and **private** clouds
  - **Software as a service** (SaaS) for applications
  - **Platform as a service** (PaaS) for entire software stack (e.g. database server)
  - **Infrastructure as a service** (IaaS) for servers or storage
  - **Load balancers** spread traffic across many servers

Open-Source Operating Systems

- Operating systems made available in source-code format rather than just binary closed-source
- Counter to the copy protection and Digital Rights Management (DRM) movement
- Started by Free Software Foundation (FSF), which has "copyleft" **GNU Public License** (GPL)
- Examples include *GNU/Linux* and *BSD UNIX* (including core of Mac OS X)

Operating System Services

- User Interface (UI)
- Program execution: load, run, end
- I/O operations
  - User programs cannot execute I/O operations directly, so the operating system must provide means to perform I/O
- File-system manipulation
- Communications – exchange of information between processes executing either on the same computer or on different systems tied together by a network.
  - Implemented via *shared memory* or *message passing*

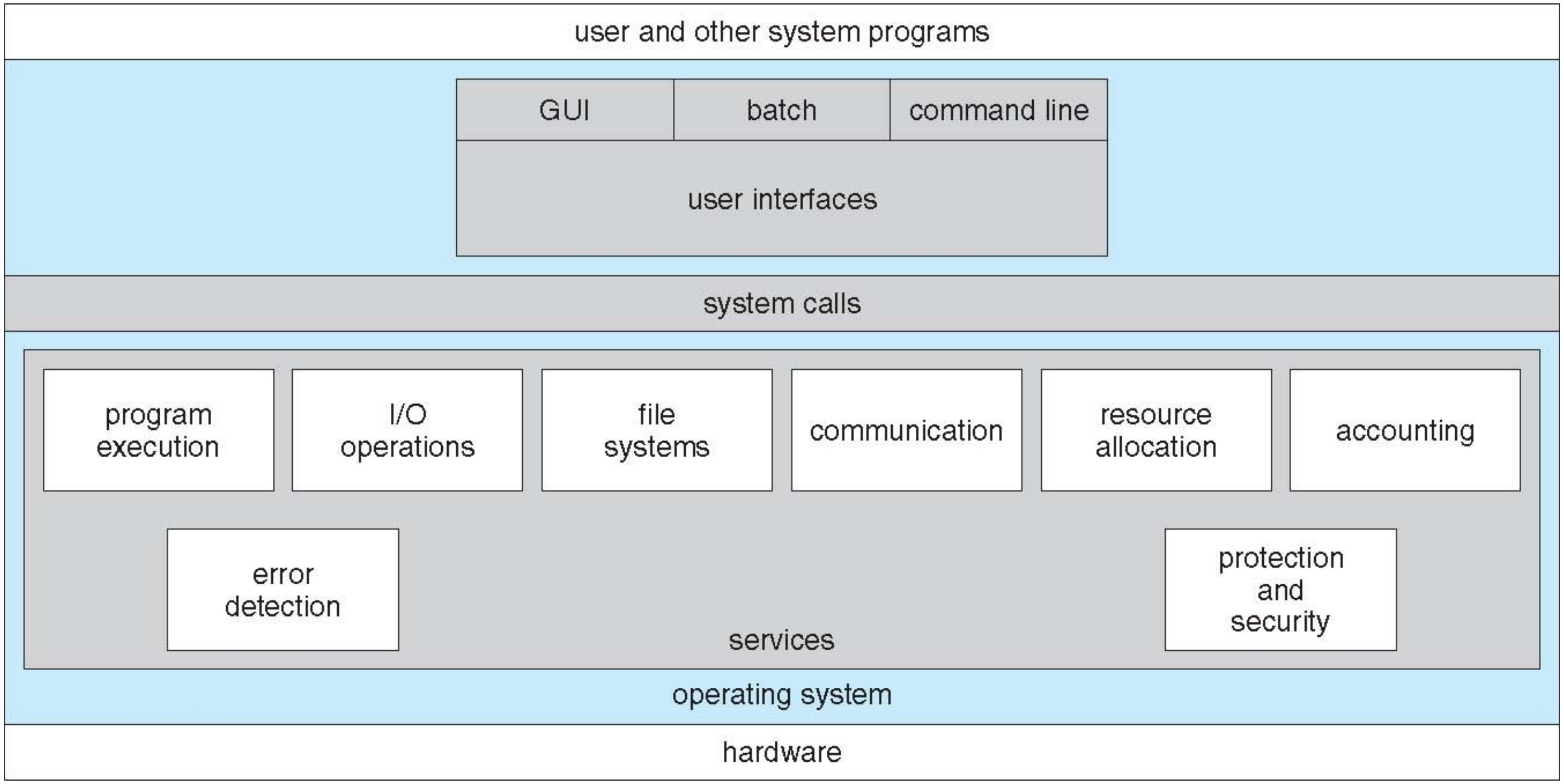
Operating System Services

- Error detection – ensure correct computing by detecting errors in the CPU and memory hardware, in I/O devices, or in user programs.
  - Should provide debugging facilities to help track down bugs

Additional functions exist not for helping the user, but rather for ensuring efficient system operations:

- Resource allocation – allocating resources to multiple users or multiple jobs running at the same time
- Accounting – keep track of and record which users use how much and what kinds of computer resources

A View of Operating System Services



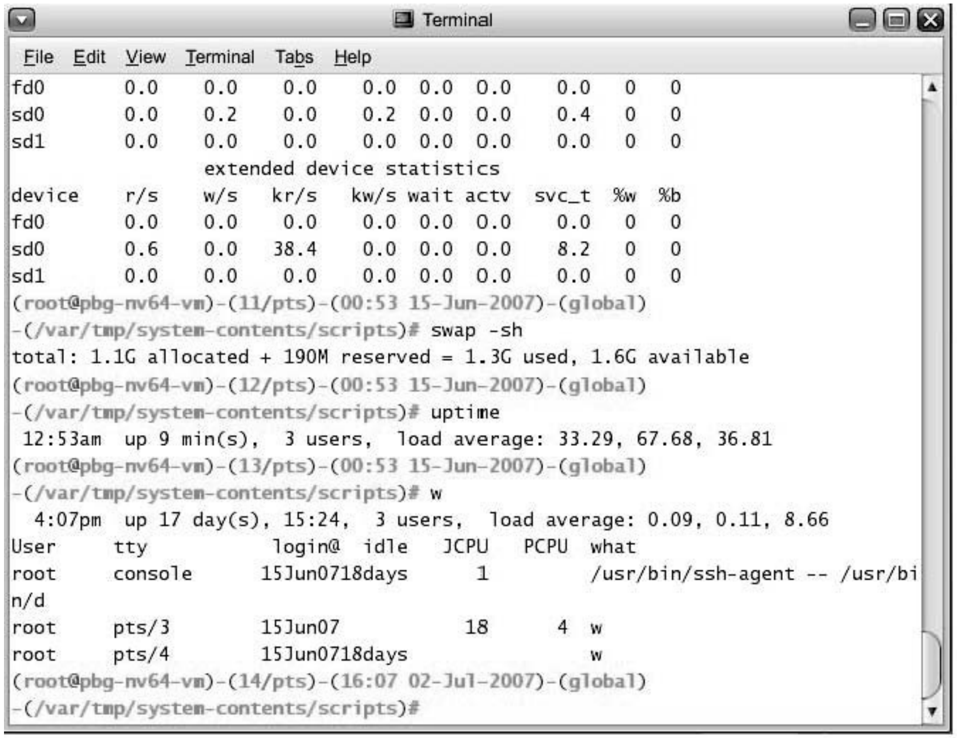
User Operating System Interface – Command-Interpreter System

- Many commands are given to the operating system by control statements typed at the keyboard (for example)
- The program that reads and interprets control statements is called variously:
  - Command-line interpreter (CLI)
  - Shell (in Unix)
- Its function is to get and execute the next command statement

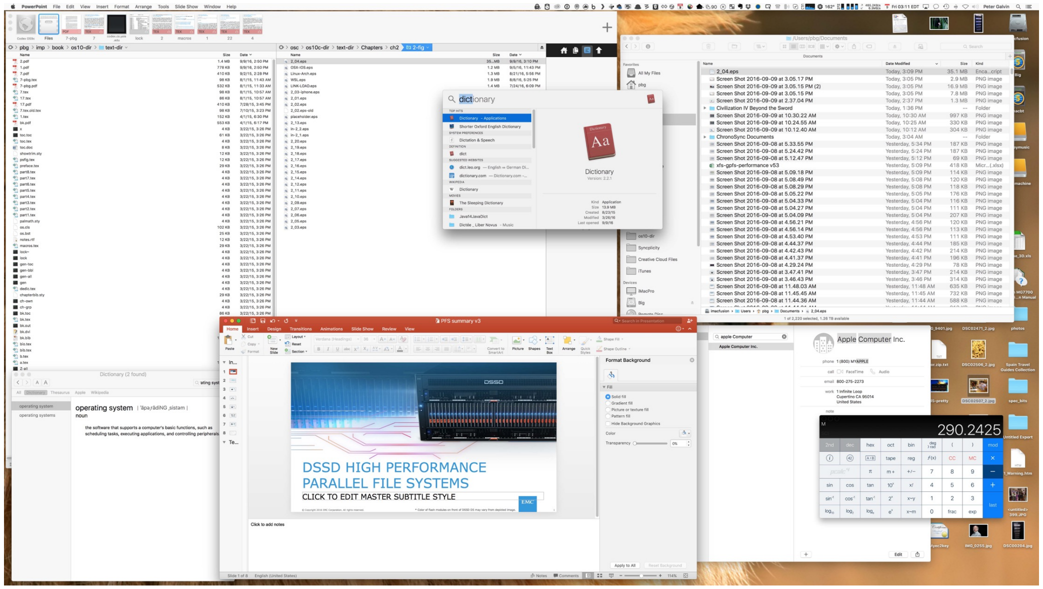
User Operating System Interface – Graphical User Interface (GUI)

- User-friendly desktop metaphor interface
  - Usually mouse, keyboard, and monitor
  - Icons represent files, programs, actions, etc
  - Invented at Xerox PARC
- Many systems now include both CLI and GUI interfaces
  - Microsoft Windows is GUI with CLI “command” shell
  - Apple Mac OS X has “Aqua” GUI interface with UNIX kernel underneath and shells available
  - Unix and Linux have CLI with optional GUI interfaces (CDE, KDE, GNOME)

Bourne Shell CLI vs. Mac OS/X GUI

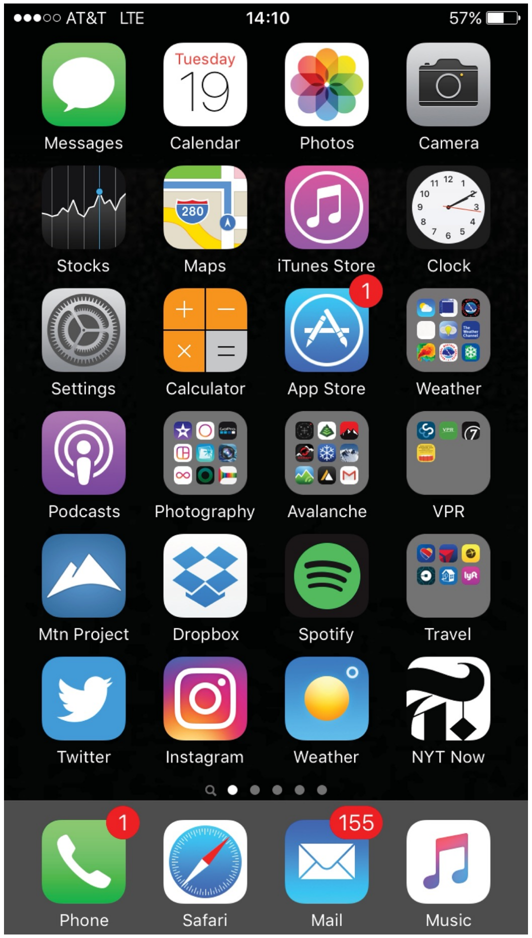


```
fd0       0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
sd0       0.0    0.2    0.0    0.2  0.0  0.0    0.4   0   0
sd1       0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
                 extended device statistics
device    r/s    w/s   kr/s   kw/s wait actv  svc_t  %w  %b
fd0       0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
sd0       0.6    0.0   38.4    0.0  0.0  0.0    8.2   0   0
sd1       0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
(root@pbg-nv64-vm)-(11/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# swap -sh
total: 1.1G allocated + 190M reserved = 1.3G used, 1.6G available
(root@pbg-nv64-vm)-(12/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# uptime
 12:53am  up 9 min(s),  3 users,  load average: 33.29, 67.68, 36.81
(root@pbg-nv64-vm)-(13/pts)-(00:53 15-Jun-2007)-(global)
-(/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/bi
n/d
root     pts/3        15Jun07            18      4  w
root     pts/4        15Jun0718days                 w
(root@pbg-nv64-vm)-(14/pts)-(16:07 02-Jul-2007)-(global)
-(/var/tmp/system-contents/scripts)#
```



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

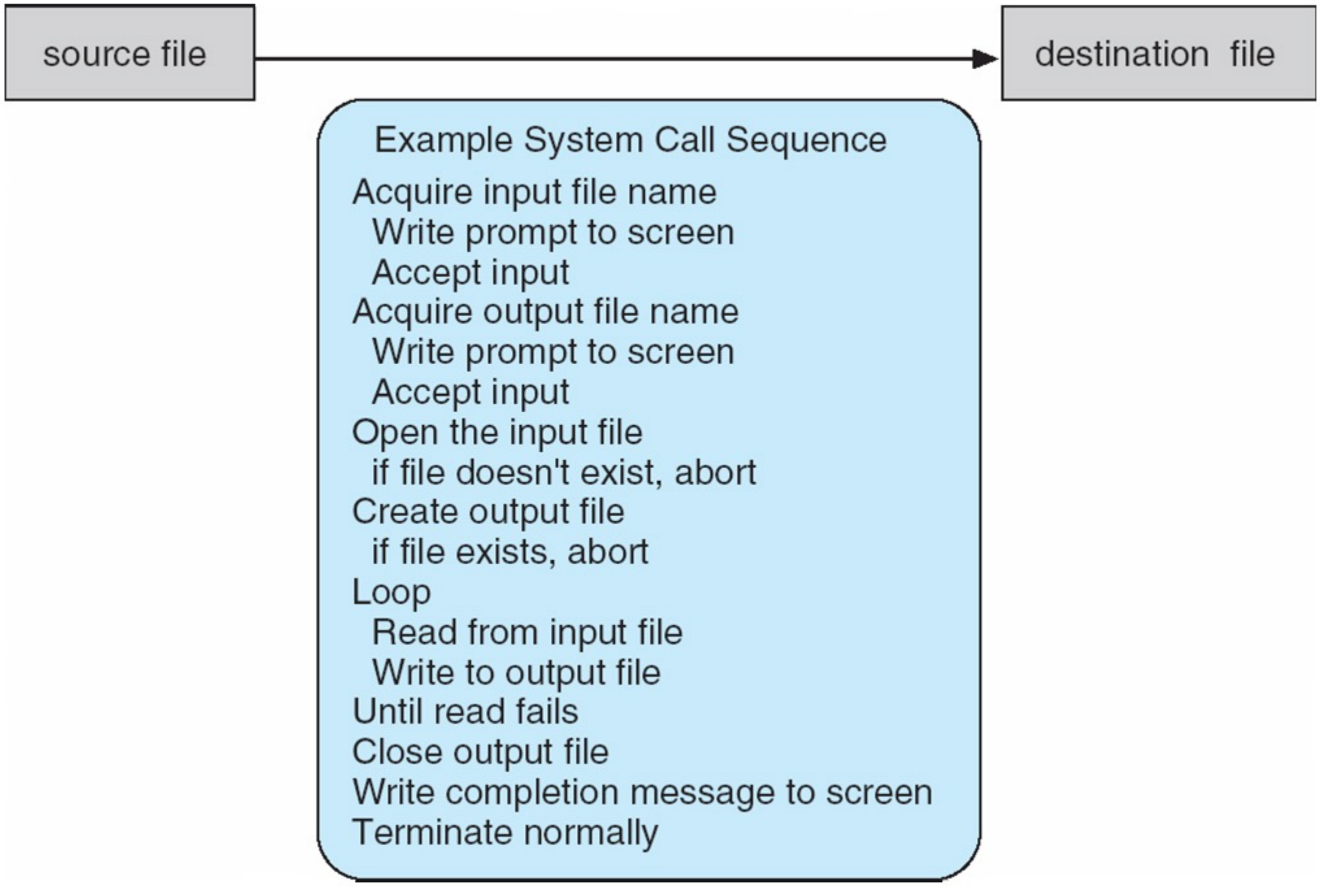
- System calls provide the interface between a running program and the operating system
- Mostly accessed by programs via a high-level **Application Program Interface (API)** rather than direct system call use
- Three most common APIs are
  - Win32 API for Windows
  - POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X)
  - Java API for the Java virtual machine (JVM)

System Calls

- Why use APIs rather than system calls?
  - Allows programs involving system calls to work on multiple platforms
  - Also, system calls are complex and difficult for programmers to use directly
- Types of system calls:
  - Process control
  - File management
  - Device management
  - Information Maintenance
  - Communications
  - Protection

Example of System Calls

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



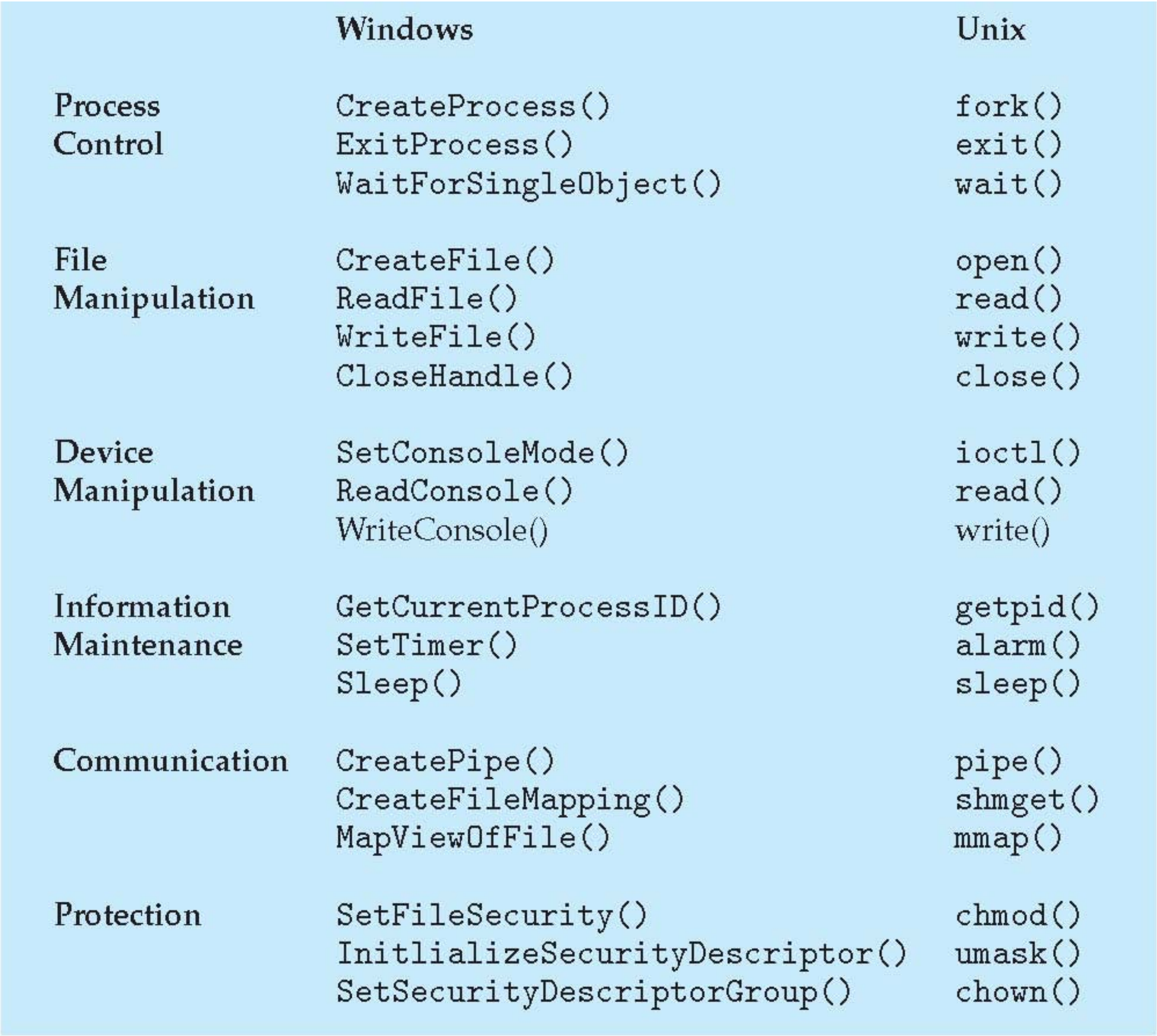
source file

destination 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
```

Examples of Windows and Unix System Calls



	Windows	Unix

Process Control	CreateProcess()
	ExitProcess()
	WaitForSingleObject()
		fork()
		exit()
		wait()

File Manipulation	CreateFile()
	ReadFile()
	WriteFile()
	CloseHandle()
		open()
		read()
		write()
		close()

Device Manipulation	SetConsoleMode()
	ReadConsole()
	WriteConsole()
		ioctl()
		read()
		write()

Information Maintenance	GetCurrentProcessID()
	SetTimer()
	Sleep()
		getpid()
		alarm()
		sleep()

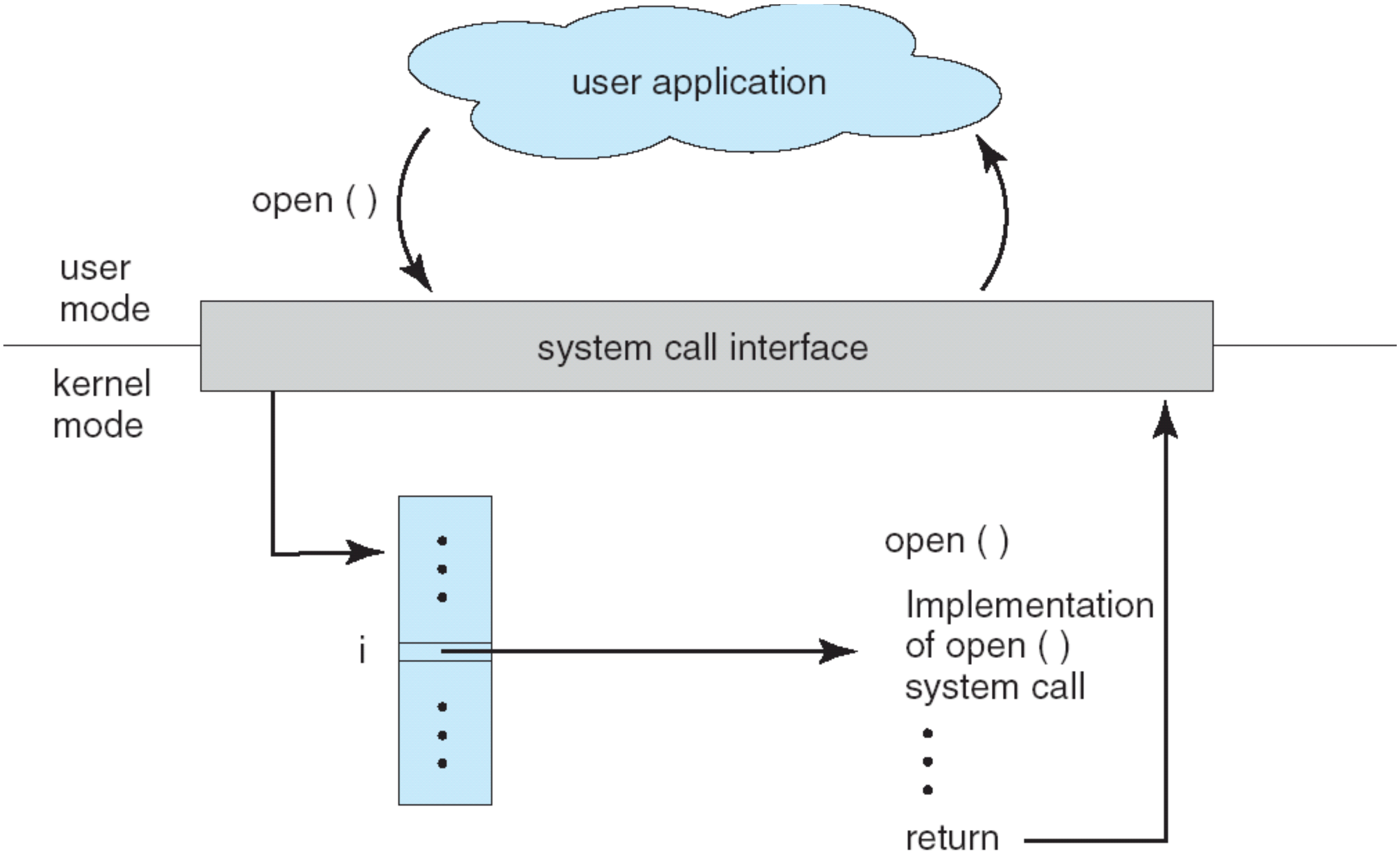
Communication	CreatePipe()
	CreateFileMapping()
	MapViewOfFile()
		pipe()
		shmget()
		mmap()

Protection	SetFileSecurity()
	InitlializeSecurityDescriptor()
	SetSecurityDescriptorGroup()
		chmod()
		umask()
		chown()

System Call Implementation

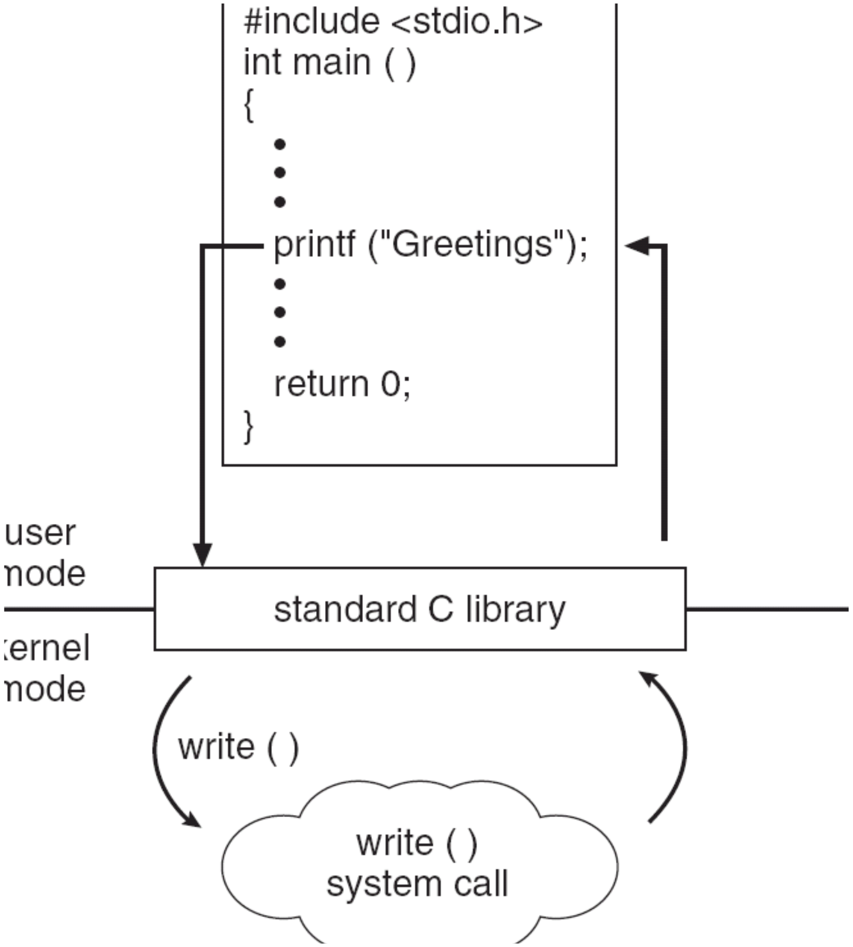
- Typically, a number associated with each system call
  - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented
  - Just needs to obey the API and understand what the OS will do as a result of the call

API – System Call – OS Relationship



Standard C Library Example

- C program invoking *printf()* library call, which calls *write()* system call

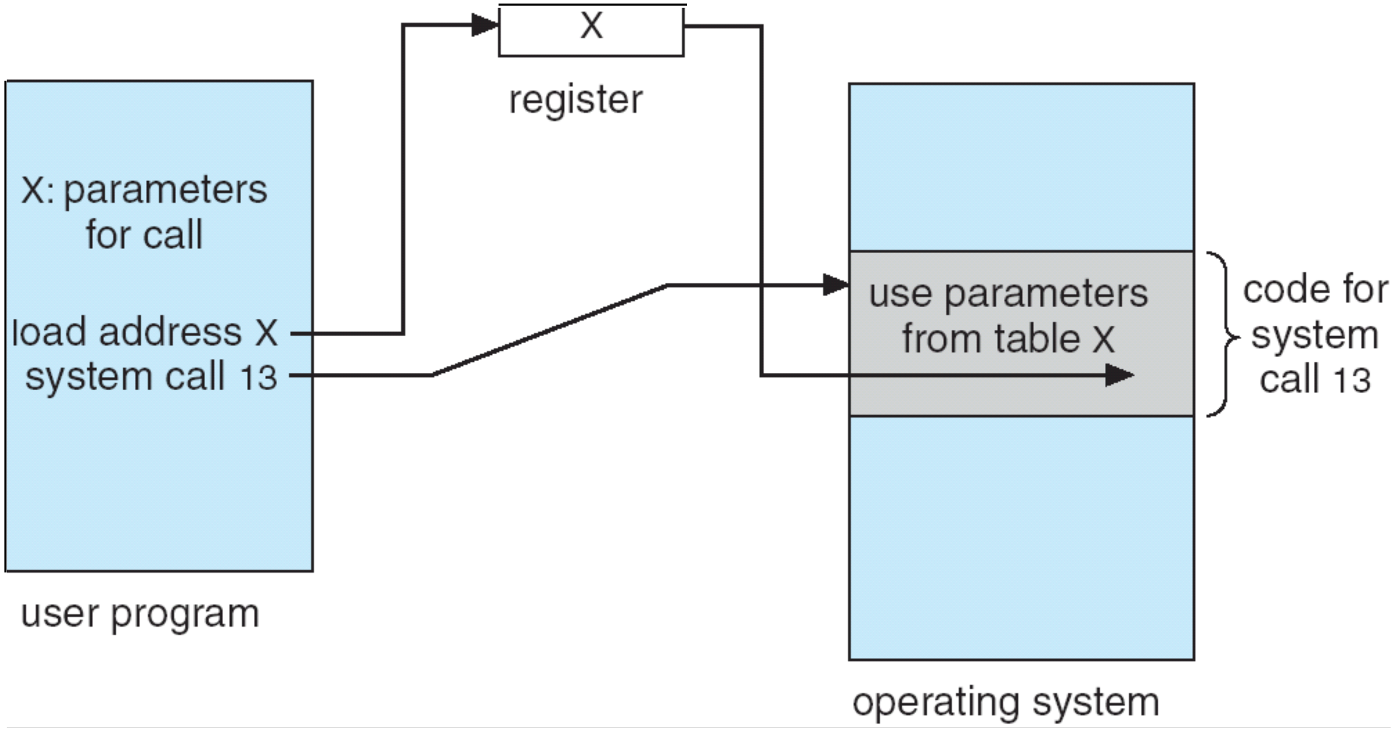


```
#include <stdio.h>
int main ( )
{
   •
   •
   •
   printf ("Greetings");
   •
   •
   •
   return 0;
}
```

System Calls – Parameter Passing

- Three general methods are used to pass parameters between a running program and the operating system:
  - Pass parameters in **registers**
  - Store the parameters in a **table** in memory
    - The table address is passed as a parameter in a register
  - Use a **stack**
    - *Push* (store) the parameters onto the stack, and *pop* them off the stack in the function
- Table and stack methods do not limit the number of parameters

Parameter Passing via Table


X
register
X: parameters
for call
load address X
system call 13
user program
use parameters
from table X
code for
system
call 13
operating system