Structures of Operating Systems

Operating Systems

School of Data & Computer Science
Sun Yat-sen University

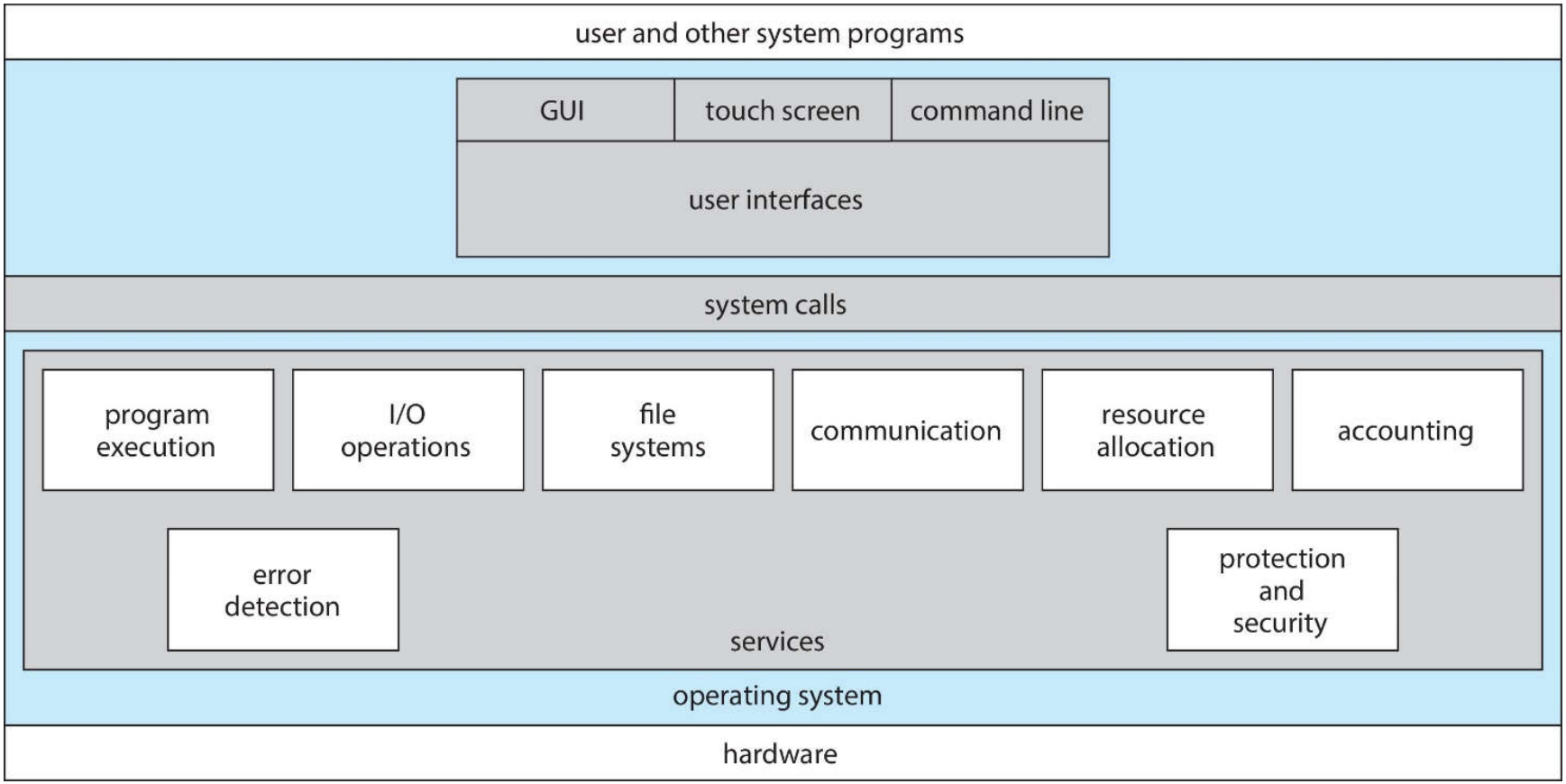
Lecture Notes: os_sysu@163.com
Instructor: Guoyang Cai
email: isscgy@mail.sysu.edu.cn

Contents

- Operating System Services
- Common Operating System Components
- System Calls and APIs
- System Programs
- Operating System Design and Implementation
- Structure/Organization/Layout of OS
  - Monolithic (one unstructured program)
  - Layered
  - Microkernel
  - Virtual Machines

Operating System Services

- A View of Operating System Services
  - These operating system services are provided for the convenience of the programmer, to make the programming task easier.
  - *A user oriented view.*



Operating System Services

- User interface (UI)
  - Command-line interface (CLI)
    - which uses text commands and a method for entering them
    - e.g., keyboard typing.
  - Batch interface
    - Commands and directives to control those commands are entered into files, and those files are interpreted and executed.
  - Graphical user interface (GUI)
    - The interface is a window system with a pointing device to direct I/O, choose from menus, and make selections and a keyboard to enter text.

Operating System Services

- System Calls
  - Program execution
    - OS capability to load a program into memory, run it, end execution, either normally or abnormally
  - I/O operations
    - User programs are not allowed to execute I/O operations directly. The OS must provide some means to perform I/O, which may involve a file or I/O device.
  - File systems
    - program capability to read, write, create, and delete files and directories
  - Communication
    - Processes may exchange information, on the same computer or between computers over a network – Implemented via shared memory or message passing.
      - InterProcess Communications (IPC)

Operating System Services

- System Calls
  - Error detection
    - Ensure correct computing by detecting errors in the CPU and memory hardware, in I/O devices, or in user programs
  - Resource allocation
    - When multiple users or multiple 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
    - Concurrent processes should not interfere with each other, and with OS system process.
    - Precautions are instituted to prevent misusing or malicious activities.

Common Operating System Components

- Operating systems components are designed and implemented for providing system services.
- *A system oriented view.*
  - Process Management
  - Main Memory Management
  - File Management
  - Mass-Storage Management
  - I/O Management
  - Networking
  - Command Line Interpreter
  - GUI
  - Protection and Security
  - Error Detection and Response
  - Accounting

Process Management

- A process is a program *in execution.*
  - A program is a *passive* entity, in static state like the contents of a file stored on disk; a process is an *active* entity, in dynamic state with a system context.
- A process needs resources to accomplish its task.
  - CPU (registers), memory, I/O devices, files.
  - Data Initialization.
  - Process termination requires reclaim of any reusable resources.
- Process is a unit of work within the system. 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 *Synchronization*.
  - providing mechanisms for process *Communication*.
  - providing mechanisms for *Deadlock* handling.

Memory Management

- The main memory (*memory* in short) is central to the operation of a modern computer system. It is a repository of quickly accessible data shared by the CPU and I/O devices.
  - It is generally the only large storage device that the CPU is able to address and access directly.
  - To execute a program, all (or part) of the instructions and data needed must be in main memory.
- The operating system is responsible for the following activities in connection with memory management:
  - *keeping track* of which parts of memory are currently being used and by whom
  - *deciding* when and which processes (or parts of processes) and data to move into and out of memory
    - optimizing CPU utilization and computer response to users
  - *Allocating* and *Deallocating* memory space as needed

File System Management

- Computers store information on several different types of physical media, each of these media has its own characteristics and physical organization.
  - The properties include *access speed*, *capacity*, *data-transfer rate*, and *access method* (sequential or random).
- A file is a collection of related information. Commonly, files represent programs (both source and object forms) and data.
- The operating system implements the abstract concept of a file by managing mass storage media and the devices that control them.
- Files are normally organized into *directories*. *Access control* (访问控制) is applied on most file systems to determine *who can access what*.
- File management activities for operating systems:
  - *Creating* and *Deleting* files
  - *Creating* and *Deleting* directories
  - supporting *Primitives* for manipulating files and directories
  - *Mapping* files onto mass storage
  - *Backing up* files on stable (nonvolatile) storage media.

Mass-Storage Management

- Usually mass-storages like HDDs and other NVM devices are 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 may hinge on disk subsystem and its algorithms.
  - Proper management of these *secondary storage* is of central importance.
- Some storage may need not be fast:
  - Tertiary storage includes optical storage, magnetic tape.
  - Varies between WORM (write-once, read-many-times) and RW (read-write)
- Mass-storage management activities for operating systems:
  - *Mounting* and *Unmounting*
  - *Free-space* management
  - storage *Allocation*
  - disk *Scheduling*
  - *Partitioning*
  - *Protection*.

Cache Management

- Caching is an important principle of computer systems. Because caches have limited size, cache management is an important design problem.
- In a multiprocessor/multicore environment, cache management becomes more complicated. In addition to maintaining internal registers, each of the CPUs also contains a local cache.
  - Cache Coherency (缓存一致性)
    - E.g., a copy of an integer A may exist simultaneously in several caches. Since the various CPUs can all execute in parallel, It must be sure that an update to the value of A in one cache is immediately reflected in all other caches where A resides. This *cache coherency* is usually a hardware issue (handled below the operating-system level).
- In a distributed environment, this situation becomes even more complex. Several copies (or replicas) of the same file can be kept on different computers. Since the various replicas may be accessed and updated concurrently, some distributed systems ensure that, when a replica is updated in one place, all other replicas are brought up to date as soon as possible.

I/O System Management

- One purpose of an OS is to hide the peculiarities of specific hardware devices from the user.
- I/O subsystem consists of several components:
  - 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
    - *Device drivers* for specific hardware devices
      - Only the device driver knows the peculiarities of the specific device to which it is assigned.

Networking

- The processors in the system are connected through a communication network.
  - Communication takes place using a protocol.
  - A networked/distributed system provides user access to various system resources.
- Access to a shared resource allows:
  - Computation speed-up
  - Increased data availability
  - Enhanced reliability

Command Line Interpreter

- The program that reads and executes commands given to the operating system
- Examples of command-line interpreters
  - Command.com (MS-DOS)
  - Shell (UNIX).
- In windows systems the interface is mouse and menu based
  - WIMP (Windows, Icons, Menu, and Pointing device).

Graphical User 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 folder
  - invented at Xerox PARC
- Systems now include both CLI and GUI interfaces:
  - Microsoft Windows is GUI with CLI “command” shell.
  - Apple macOS X as “Aqua” GUI with UNIX kernel underneath and shells available.
  - Solaris is CLI with optional GUIs (Java Desktop, KDE).
- Linux is CLI only, without GUI in kernel.
  - Gnome (GNU Network Object Model Environment) and KDE (King Desktop Environment) are graphical environments developed under X-protocol.

Protection and Security

- Protection
  - any mechanism for controlling access of processes or users to both system and user resources
- Security
  - defense of the system against internal and external attacks, with huge range, including:
    - Denial-of-service
    - Worms
    - Viruses
    - Identity theft
    - Theft of service
    - …

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 is 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 (权限升级) allows user to change to effective ID with more rights.

Error Detection and Response

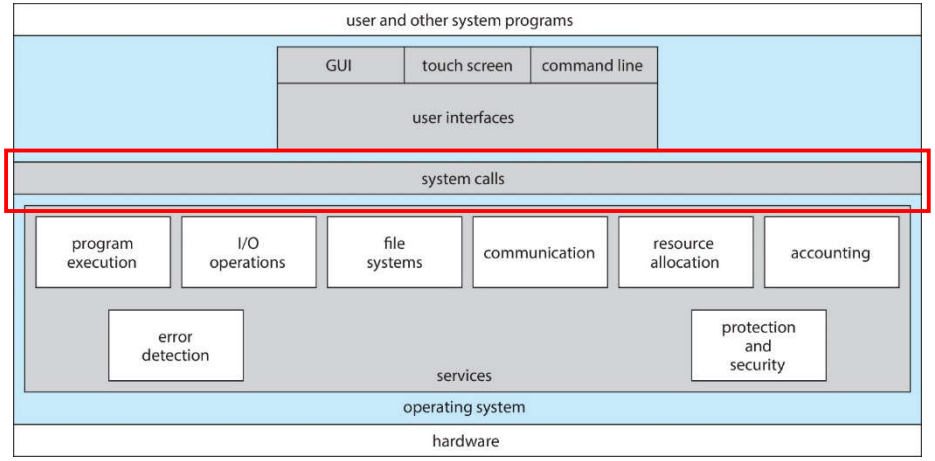
- Error Detection
  - Internal and external hardware errors
    - memory error
    - device failure
  - Software errors
    - arithmetic overflow
    - access forbidden memory locations
- Error Response
  - simply report error to the application
  - retry the operation
  - abort the application

Accounting

- Accounting keeps track of and records which users use how much and what kinds of computer resources.
- Accounting:
  - collect statistics on resource usage
  - monitor performance (e.g., response time)
  - used for system parameter tuning to improve performance
  - useful for anticipating future enhancements
  - used for billing users (on multi-user systems)

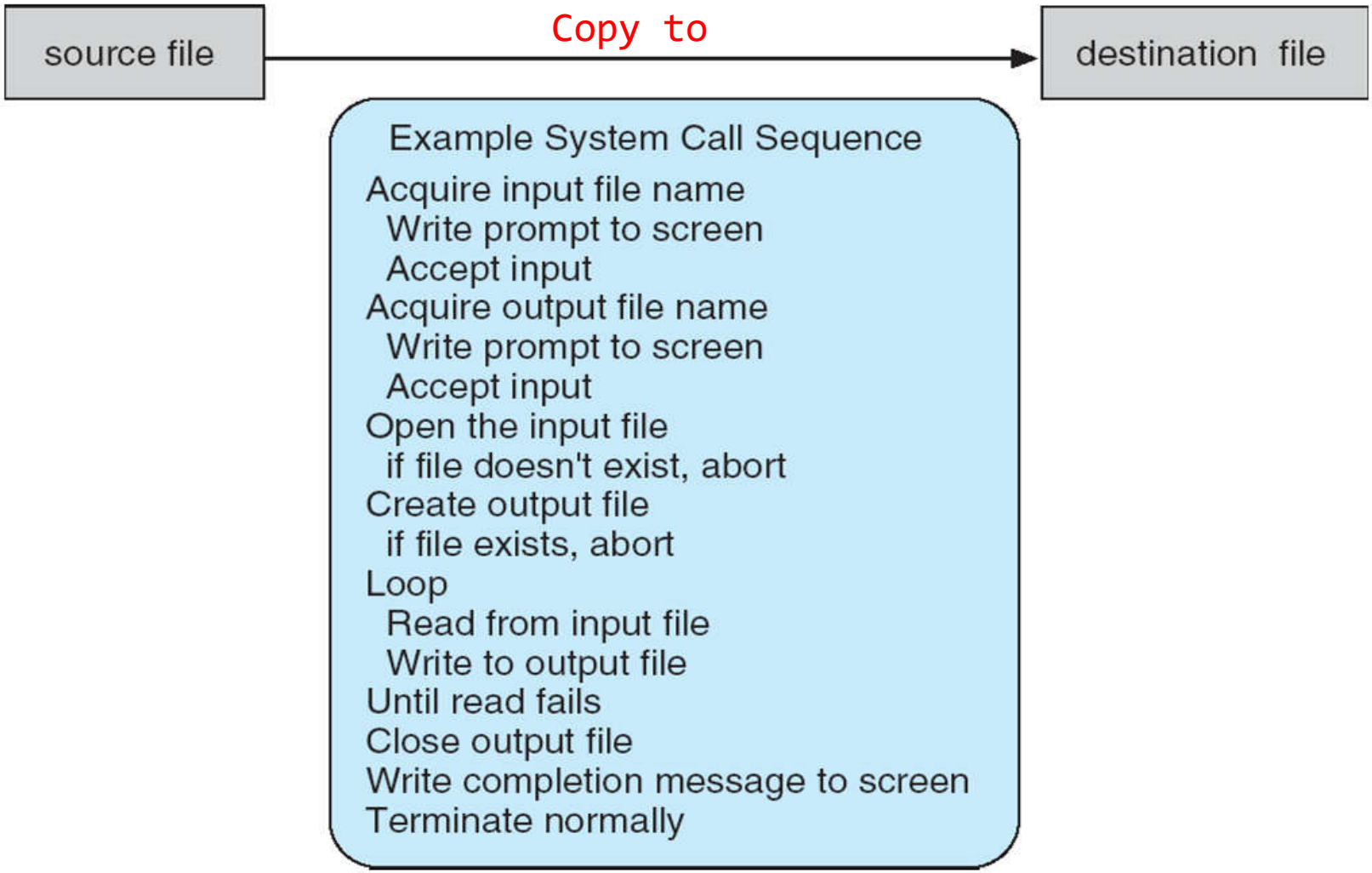
System Calls and APIs

- Systems calls provide *programming interface* to the services provided by the OS.
  - Typically written in a high-level language (C/C++).
  - Mostly accessed by programs via a high-level Application Program Interface (API), rather than through direct system calls.
- Three most common APIs:
  - Win32 API for Windows
  - POSIX API for POSIX-based systems
    - including virtually all versions of UNIX, Linux, and mac OS X.
    - POSIX: Portable Operating System Interface.
  - Java API for the Java Virtual Machine (JVM).



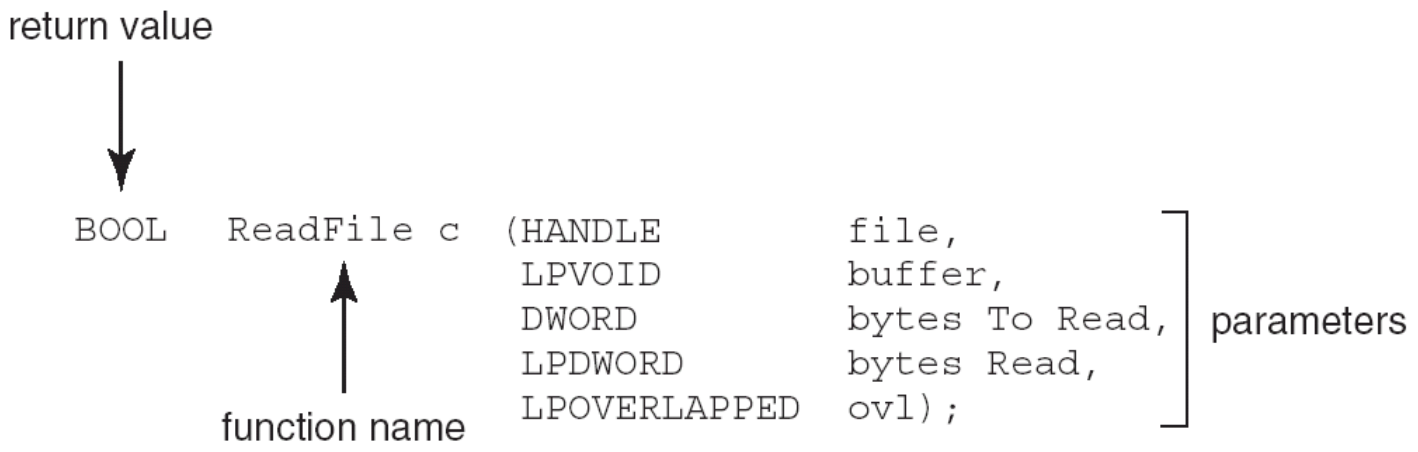
System Calls and APIs

- Example of Systems Call.



System Calls and APIs

- Example of Standard API
  - Consider the ReadFile() function in the Win32 API.
    - function for reading from a file.



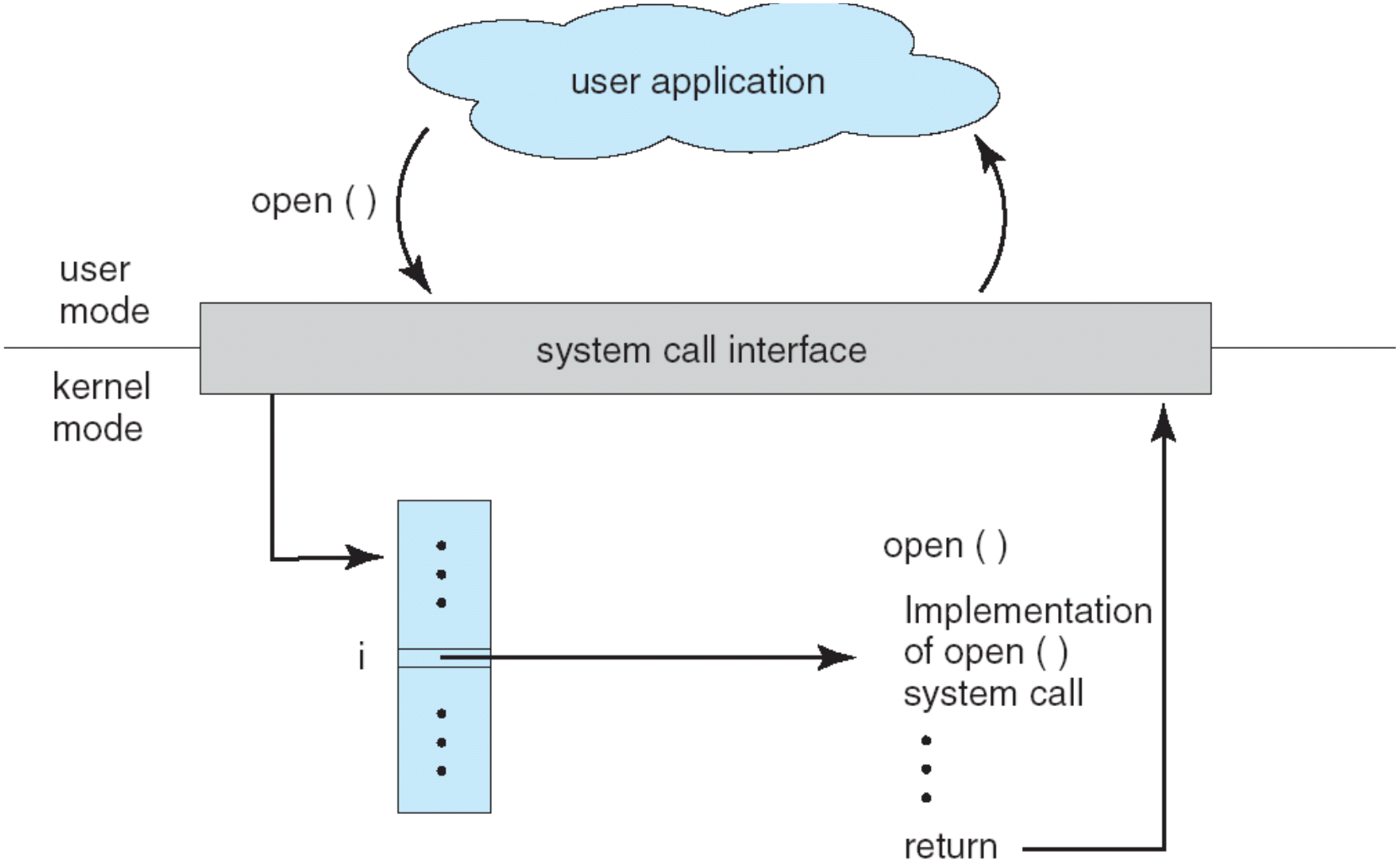
  - A description of the parameters passed to ReadFile()
    - HANDLE file – the file to be read
    - LPVOID buffer – a buffer where the data will be read into and written from
    - DWORD bytesToRead – the number of bytes to be read into the buffer
    - LPDWORD bytesRead – the number of bytes read during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

System Calls and APIs

- System Call Implementation
  - Typically a number is associated with each system call.
    - System-call interface maintains a table indexed according to system-call numbers.
  - The system call interface invokes intended system call in 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 call
    - Details of OS interface are hidden from programmer by API since managed by run-time support library (set of functions built into libraries included with compiler).

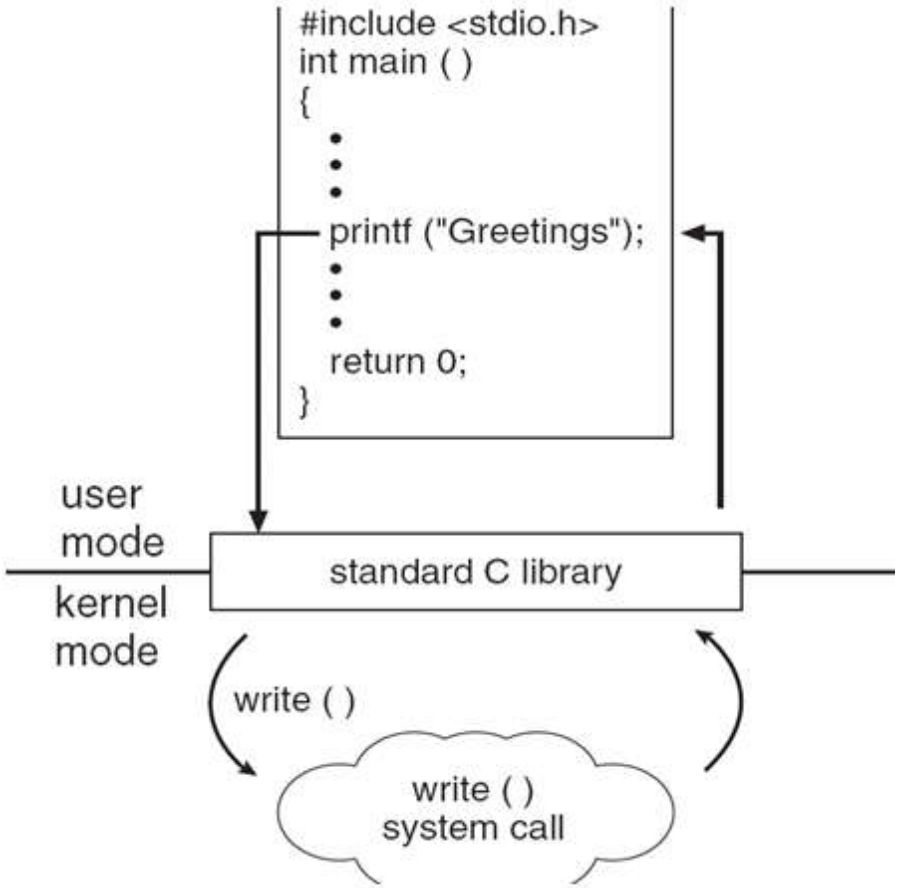
System Calls and APIs

- Relationship of API, System Call and OS.



System Calls and APIs

- Standard C Library
  - Example: C program invoking printf() library call, which calls write() system call.

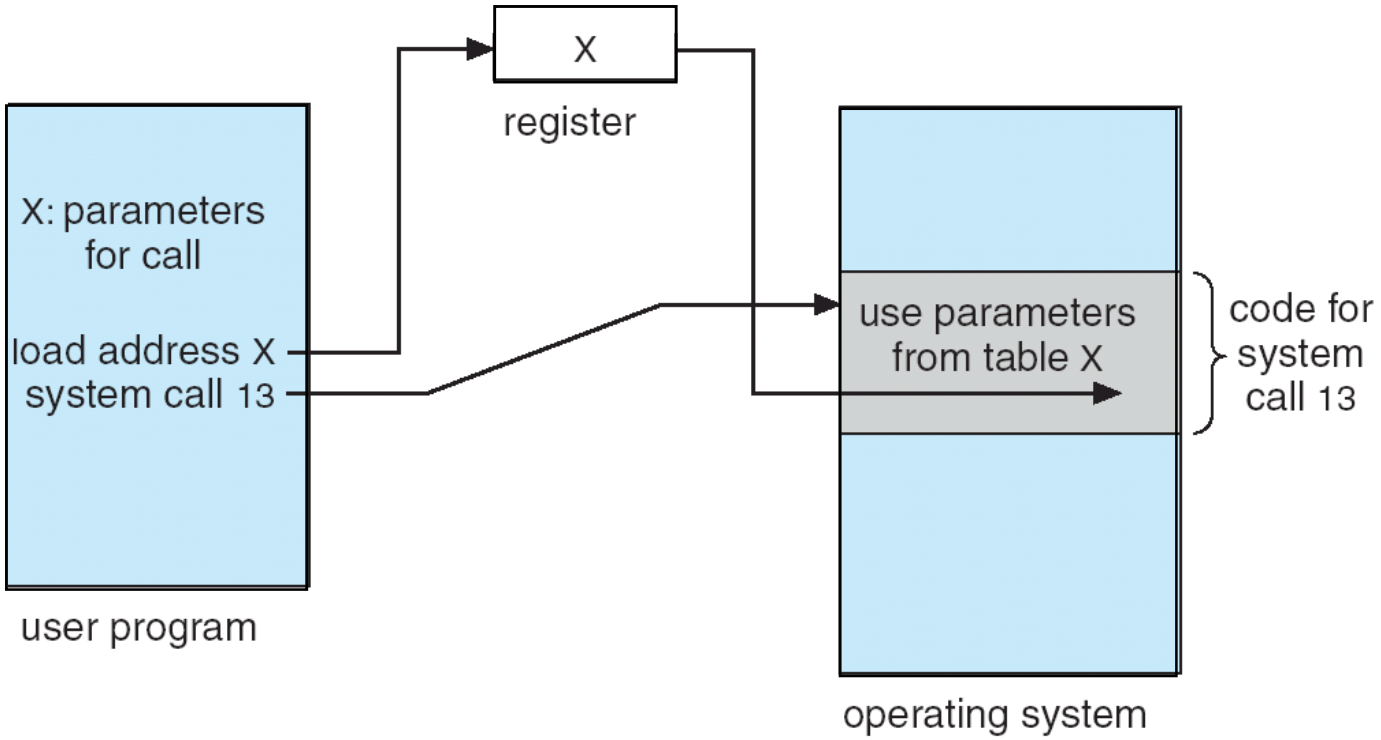


System Calls and APIs

- System Call Parameter Passing
  - Often, more information is required than simply identity of desired system call, but exact type and amount of information vary according to the OS and the call.
  - Three general methods used to pass parameters to the OS:
    - simplestly pass the parameters in *registers*.
      - In some cases, there 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 is 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.

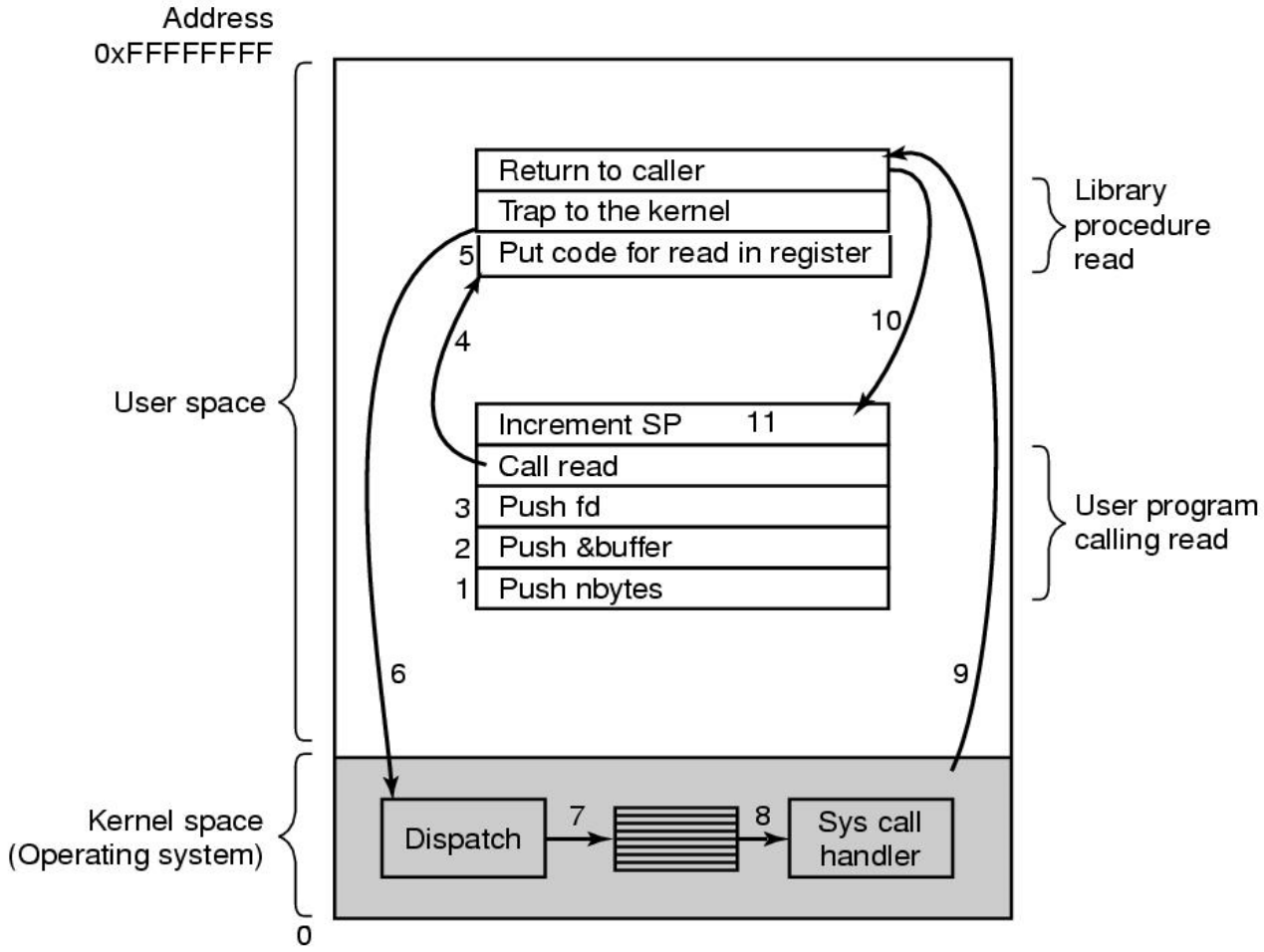
System Calls and APIs

- System Call Parameter Passing
  - Passing via Table.



System Calls and APIs

- System Call Parameter Passing
  - Passing via Stack.
    - steps 1-11 in making the system call read(fd, buffer, nbytes)



System Calls and APIs

- Examples of POSIX System Calls
  - POSIX: Portable Operating System Interface of UNIX, IEEE 1003.X 1992-1998.

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

System Calls and APIs

- Examples of POSIX System Calls
  - POSIX: Portable Operating System Interface of UNIX, IEEE 1003.X 1992-1998.

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing, or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

System Calls and APIs

- Win32 API calls roughly correspond to UNIX calls.

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

System Programs

- System programs provide a convenient environment for program development and execution. Some of them are simply user interfaces to system calls; others are considerably more complex.
- Most users' view of the operating system is defined by system programs, not the actual system calls.
- System programs provide:
  - File and directories management/modification
  - Status information
  - Programming language support
  - Program loading and execution
  - Communications
  - Application programs.

System Programs

- File and directories management
  - Create
  - Delete
  - Copy
  - Rename
  - Print
  - Dump
  - List
- File modification
  - Text editors to create and modify files.
  - Special commands to search contents of files or perform transformations of the text.

System Programs

- Status information
  - Some system programs ask the system for information
    - Date
    - Time
    - Free memory
    - Free disk space
    - Number of users.
  - Others provide detailed performance, logging, and debugging information.
  - Typically, these system programs format and print the required status information to the terminal or other output devices.
  - Some systems implement a registry used to store and retrieve configuration information.

System Programs

- Programming language support
  - Compilers
  - Assemblers
  - Debuggers
  - Interpreters.
- Program loading and execution
  - Absolute loaders
  - Relocatable loaders
  - Linkage editors
  - Overlay loaders
  - Debugging systems.

System Programs

- Communications
  - Provide the mechanism for creating virtual connections among
    - Processes
    - Users
    - Computer systems.
  - Network communications 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.
- Application programs Examples
  - Database systems
  - Compilers
  - Web browsers
  - Word processors
  - Text formatters
  - Spreadsheets
  - Plotting and statistical-analysis packages
  - Games.

Operating System Design and Implementation

- Design and Implementation of OS are not “solvable”, but some approaches have proven successful.
  - Internal structure of different operating systems can vary widely.
  - affected by choice of hardware, type of system.
  - start by defining goals and specifications.
- User goals and system goals
  - User goals
    - Operating system should be convenient to use, easy to learn, reliable, safe, and fast.
  - System goals
    - Operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient.

Operating System Design and Implementation

- Separation of Mechanisms and Policies
  - The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later.
    - Policy: What will be done?
    - Mechanism: How to do it?
  - Example.
    - There is an abstract priority queue. We need to support mechanisms for:
      - Insert/Delete items at start.
      - Insert/Delete items at end.
      - Know length of the queue.
    - Body/code of queue can be implemented in different ways.
    - Policies can be for example FIFO, LIFO – should be decided by queue user.

Operating System Design and Implementation

- System Implementation
  - traditionally written in assembly language, most parts of operating systems can now be written in higher-level languages
  - code written in a high-level language:
    - written faster
    - more compact
    - easier to understand and debug
  - An OS is far easier to port (move to some other hardware) if it is written in a high-level language.

Operating System Design and Implementation

- Open-Source Operating Systems
  - 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, BSD UNIX, and many more.

Operating System Design and Implementation

- Operating System Debugging
  - Debugging is finding and fixing errors, or bugs.
  - *Brian Kernighan*'s Law
    - "Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."
    - *Brian W. Kernighan* & *Dennis M. Ritchie* invented The C Programming Language in 1978.
  - An OS generates log files containing error information.
    - Failure of an application can generate core dump file capturing memory of the process.
    - Operating system failure can generate crash dump file containing kernel memory.
  - DTrace tool in Solaris, FreeBSD, Mac OS X
    - Probes fire when code is executed, capturing state data and sending it to consumers of those probes.

Operating System Design and Implementation

- Operating System Generation & Boot
  - Operating Systems are designed to run on any of a class of machines; OS must be configured for each specific computer site.
    - SYSGEN (System Generation) program obtains information concerning the specific configuration of the hardware system.
  - Booting
    - Operating system must be made available to hardware so the hardware can start it loading the kernel.
  - Bootstrap program (引导程序)
    - Bootstrap program is a small piece of code stored in ROM or EPROM (known as firmware).
    - When power-up or reboot, execution of bootstrap program starts at a fixed memory location.
      - Sometimes two-step process where boot block at fixed location loads bootstrap loader
    - The bootstrap program locates the kernel, load it into memory, and start its execution.
      - Initializes all aspects of system