

CS343: Operating System

OS Mode and Services

Lect05 : 7th Aug 2023

Dr. A. Sahu

Dept of Comp. Sc. & Engg.

Indian Institute of Technology Guwahati

Outline

- OS Structure
 - Multiprogramming, Time Sharing, CPU Scheduling, Swapping
- OS Mode
- OS Services

OS Operation Mode

Interrupt, Kernel Mode & User Mode

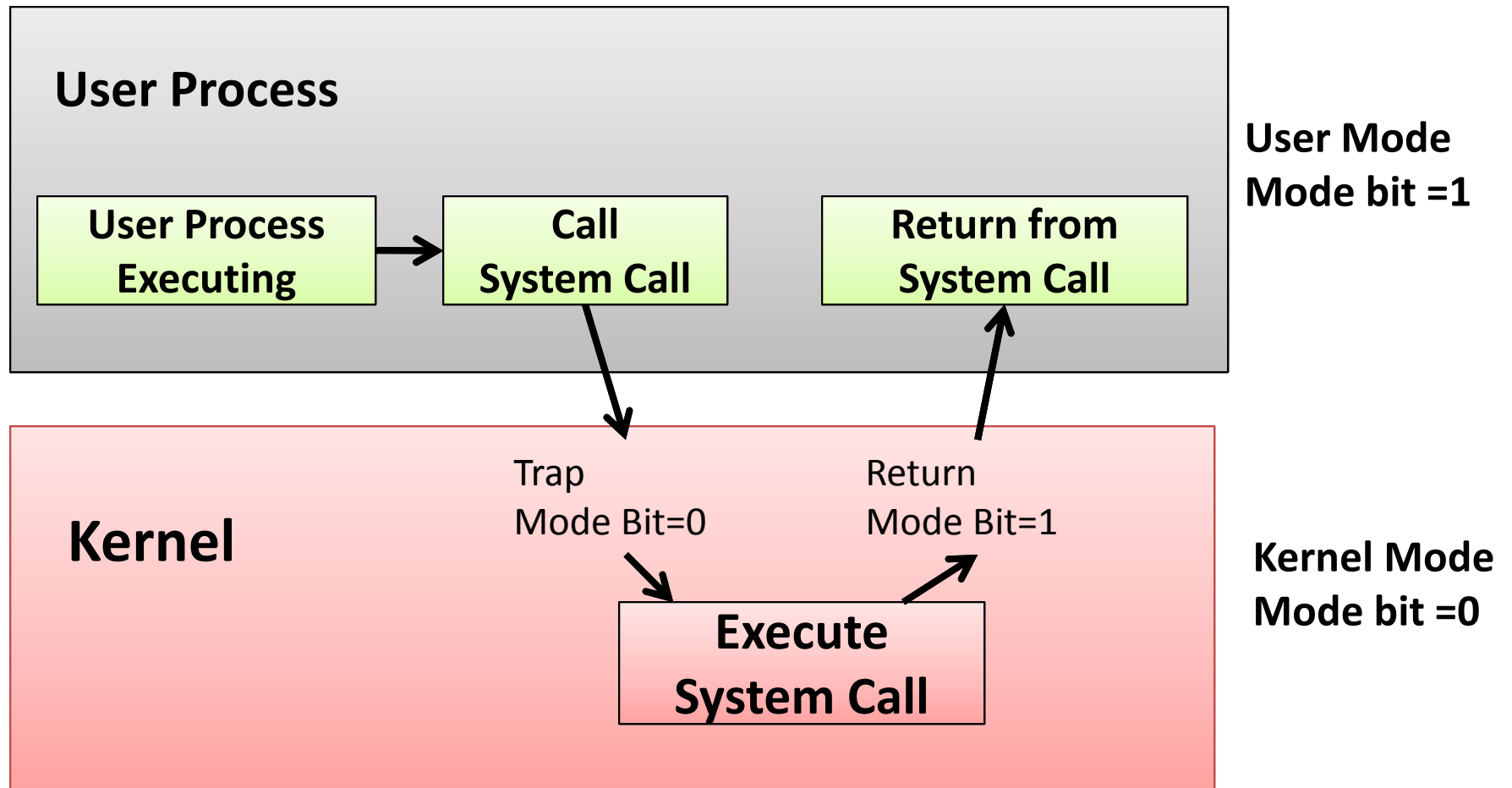
Operating-System Operations

- **Interrupt driven** (H/W and S/W)
 - H/W interrupt by one of the devices
 - S/W interrupt (**exception** or **trap**):
 - Software error (e.g., division by zero)
 - Request for OS service
 - Other process problems include
 - infinite loop
 - processes modifying each other or the operating system

Operating-System Operations (cont.)

- **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
- Increasingly CPUs support multi-mode operations
 - i.e. **virtual machine manager (VMM)** mode for guest VMs

Transition from User to Kernel Mode



Transition from User to Kernel Mode

- Timer to prevent infinite loop / process hogging resources
 - Timer is set to interrupt after some time period
 - Keep a counter decremented by the physical clock.
 - OS set the counter (privileged instruction)
 - When counter zero generate an interrupt
 - Set up before scheduling process to regain control or terminate program that exceeds allotted time

Installing Linux on 8085 or 8086

- Is it possible to install Linux OS on top of 8085 or 8086 based system ?
- **No**
- Because it don't support mode bit
 - Kernel Mode or user mode bit
 - i386,i586,i686.....in short ix86 support mode bit

Services of Operating System

Operating System Services

- OS provide
 - An environment for execution of programs
 - And services to programs and users
- Services
 - One set of OS services provides functions that are *helpful to the user*
 - Another set of OS functions exists for *ensuring the efficient operation of the system* itself via resource sharing

OS Services: provides functions that are helpful to the user

- **User interface** - Almost all OSs 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
 - 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

OS Services: provides functions that are helpful to the user

- **File-system manipulation** - The file system is of particular interest.
 - Programs need to 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
 - Communications may be via shared memory or through message passing (packets moved by the OS)

OS Services: provides functions that are helpful to the user

- **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

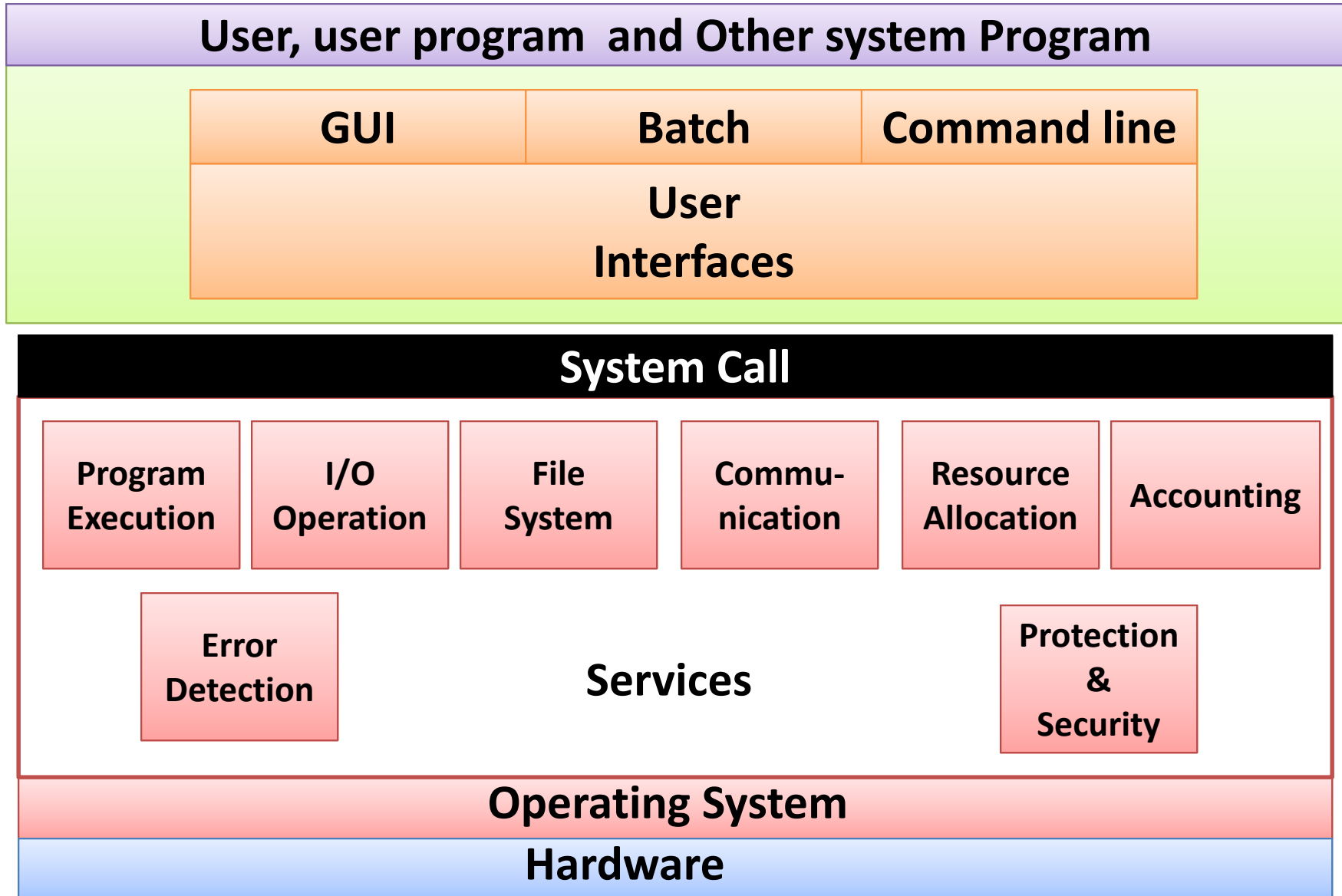
OS Services: provides functions that ensure efficient operation of System

- **Resource sharing and 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

OS Services: provides functions that ensure efficient operation of System

- **Protection and security –**
 - The owners of information stored in a multiuser or networked computer system may want to control use of that information
 - 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 OS Services

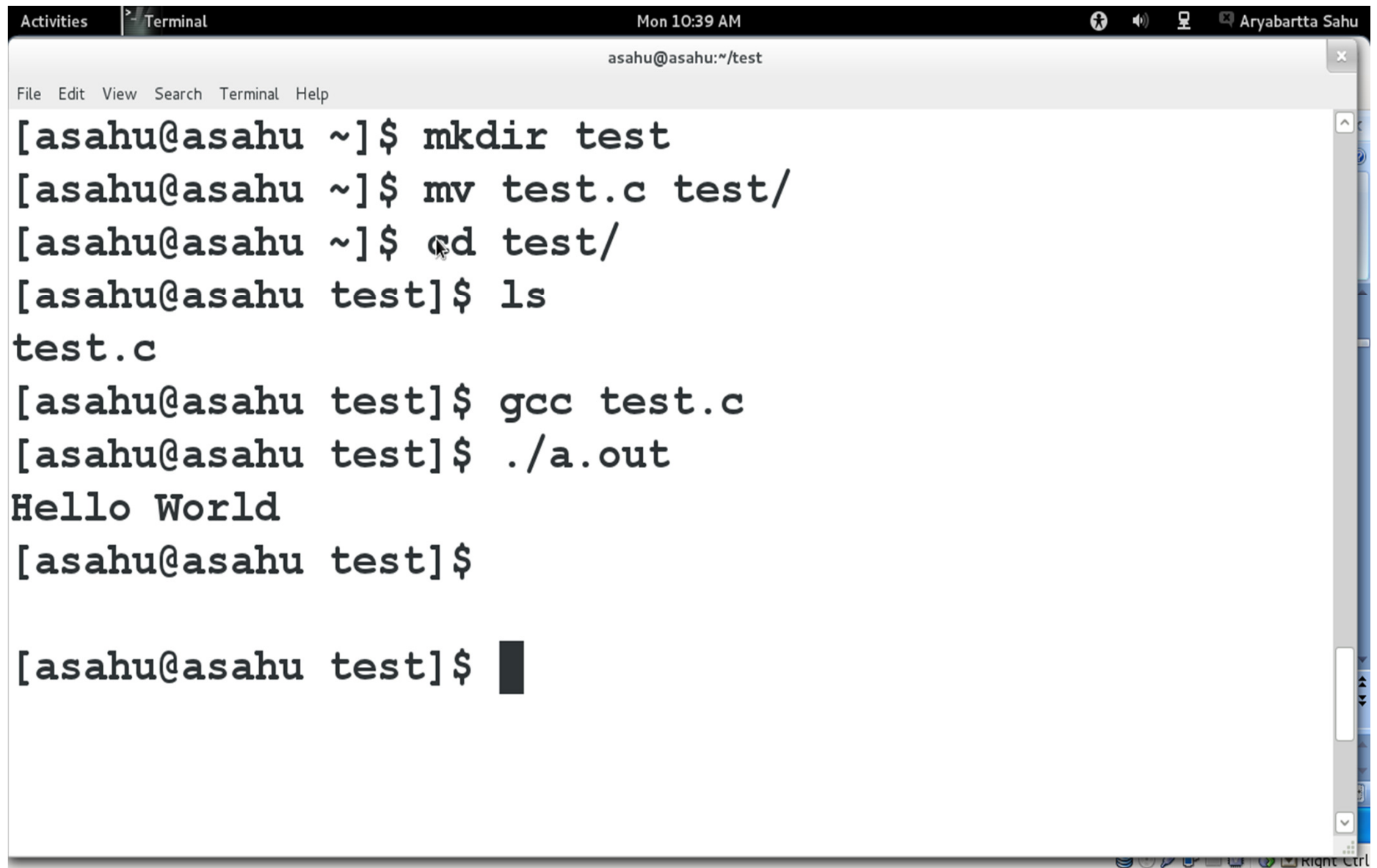


OS User Interface - CLI

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

Linux Shell: Command Interpreter



The image shows a terminal window titled "Terminal" with the user "asahu" at host "asahu". The window displays a series of commands and their outputs. The commands executed are: `mkdir test`, `mv test.c test/`, `cd test/`, `ls`, `gcc test.c`, and `./a.out`. The output of `ls` is `test.c`, and the output of `./a.out` is `Hello World`. The terminal window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The status bar at the bottom shows system icons and the text "Right Ctrl".

```
Activities | Terminal | Mon 10:39 AM | Aryabartta Sahu
asahu@asahu:~/test
File Edit View Search Terminal Help
[asahu@asahu ~]$ mkdir test
[asahu@asahu ~]$ mv test.c test/
[asahu@asahu ~]$ cd test/
[asahu@asahu test]$ ls
test.c
[asahu@asahu test]$ gcc test.c
[asahu@asahu test]$ ./a.out
Hello World
[asahu@asahu test]$
[asahu@asahu test]$
```

OS User Interface - GUI

- Graphical User Interfaces
- 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**

OS User Interface - GUI

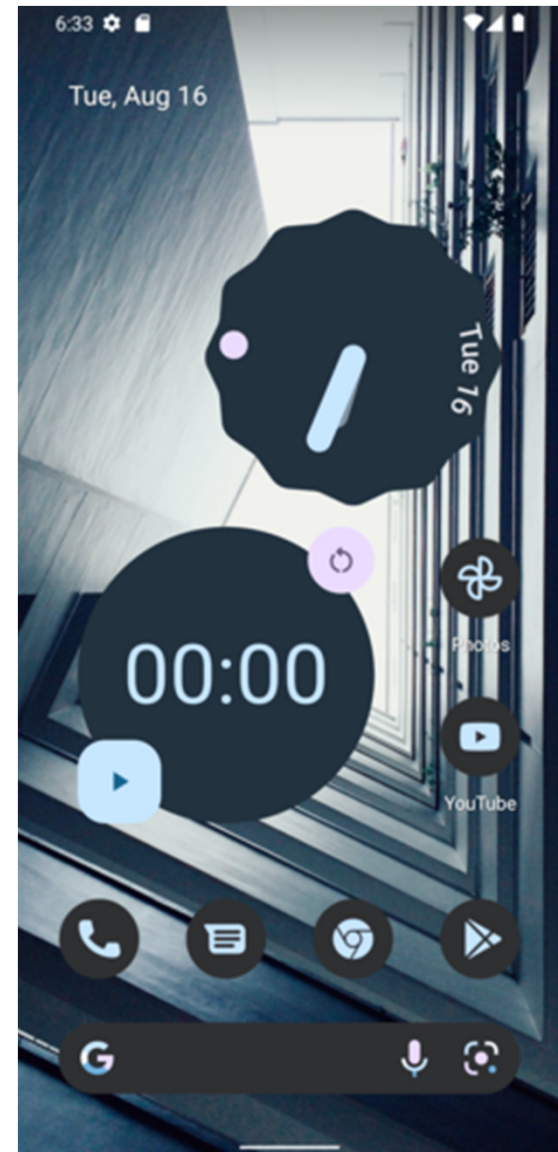
- Many systems now include both CLI and GUI interfaces
 - Microsoft **Windows** is GUI with CLI “command” shell
 - Apple Mac OS X is “**Aqua**” GUI interface with UNIX kernel underneath and shells available
 - Unix and Linux have CLI with optional GUI interfaces (CDE, **KDE, GNOME**)

Mac Book OS 13 GUI



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.
- **Android 13 Tiramisu**



System Calls

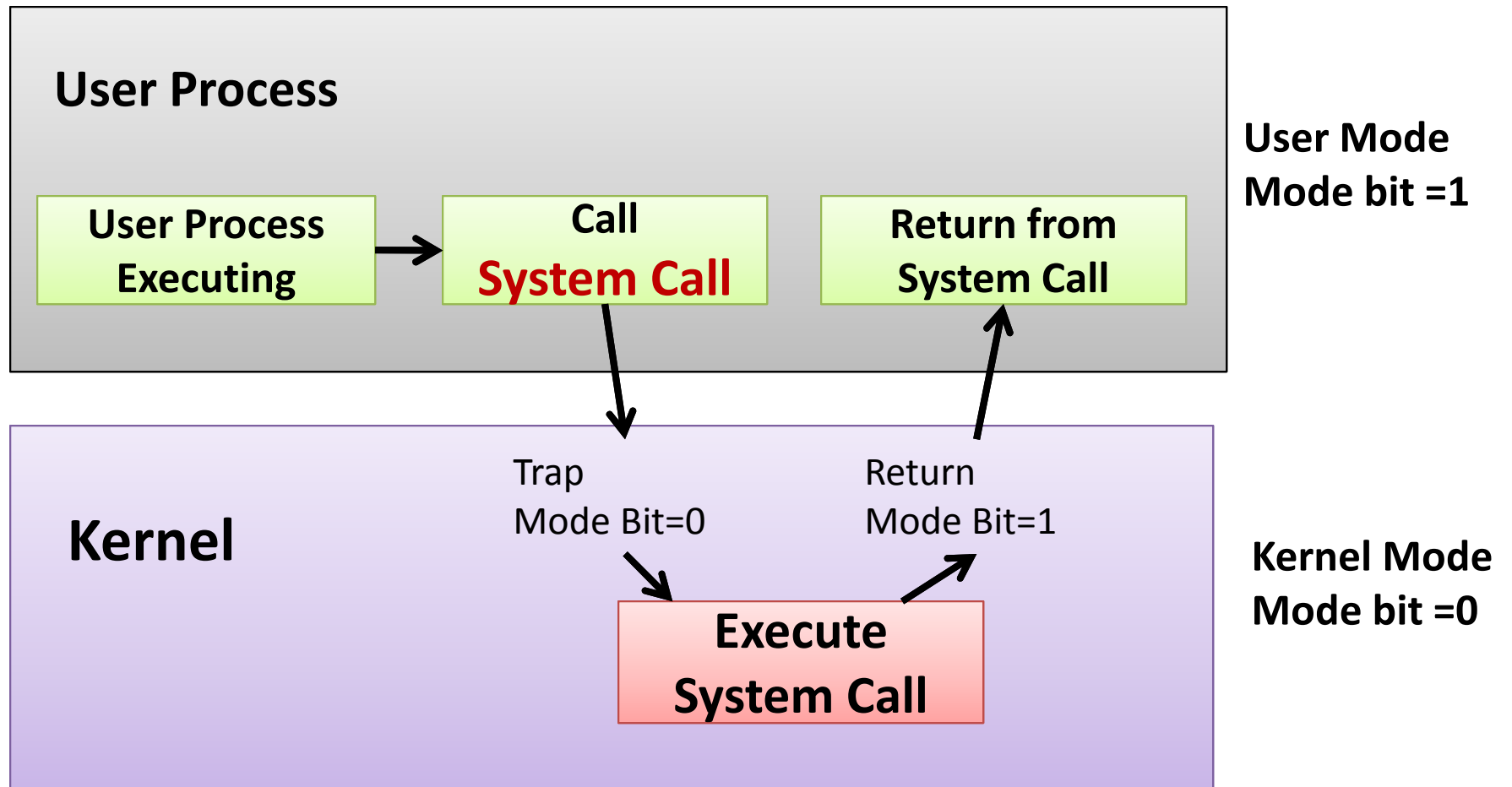
- Programming interface to the services provided by the OS
 - Typically written in a high-level language (C or C++)
- Mostly accessed by programs
 - Via a high-level **Application Programming Interface (API)**
 - rather than direct system call use
- Three most common APIs are
 - **Win32 API** for Windows
 - **POSIX API** for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X),
 - **Java API** for the Java virtual machine (JVM)

Note that the system-call names used throughout this course are generic

Operating-System Operations Modes

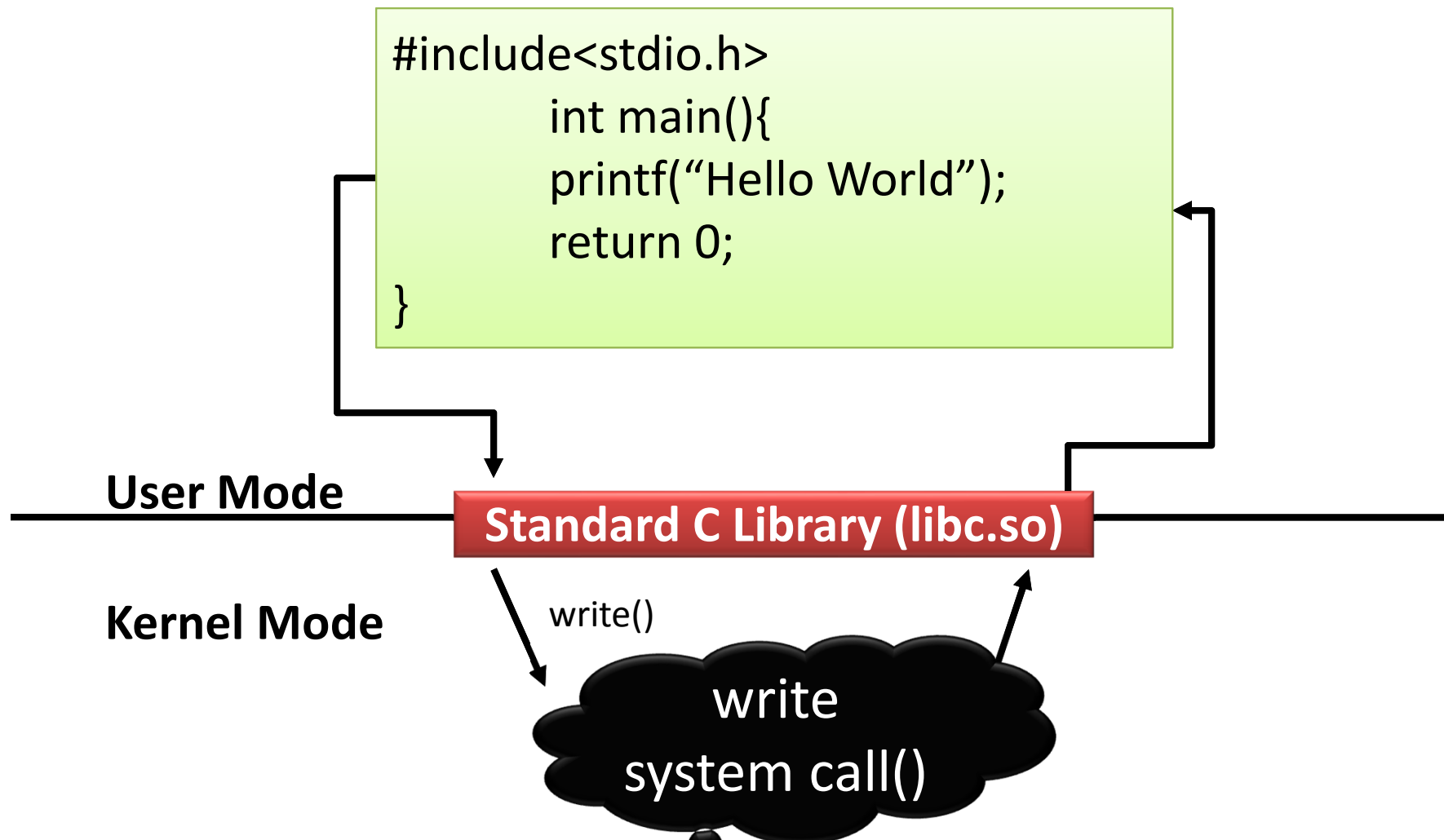
- **Dual-mode** operation allows OS to protect itself and other system components
 - **User mode** and **kernel mode**
 - **Mode bit** provided by hardware
 - Provides ability to distinguish when system is running user code or kernel code
 - Some instructions designated as **privileged**, only executable in **kernel mode**
 - **System call** changes mode to kernel, return from call resets it to user
- Increasingly CPUs support multi-mode operations
 - i.e. **virtual machine manager (VMM)** mode for guest VMs

Transition from User to Kernel Mode



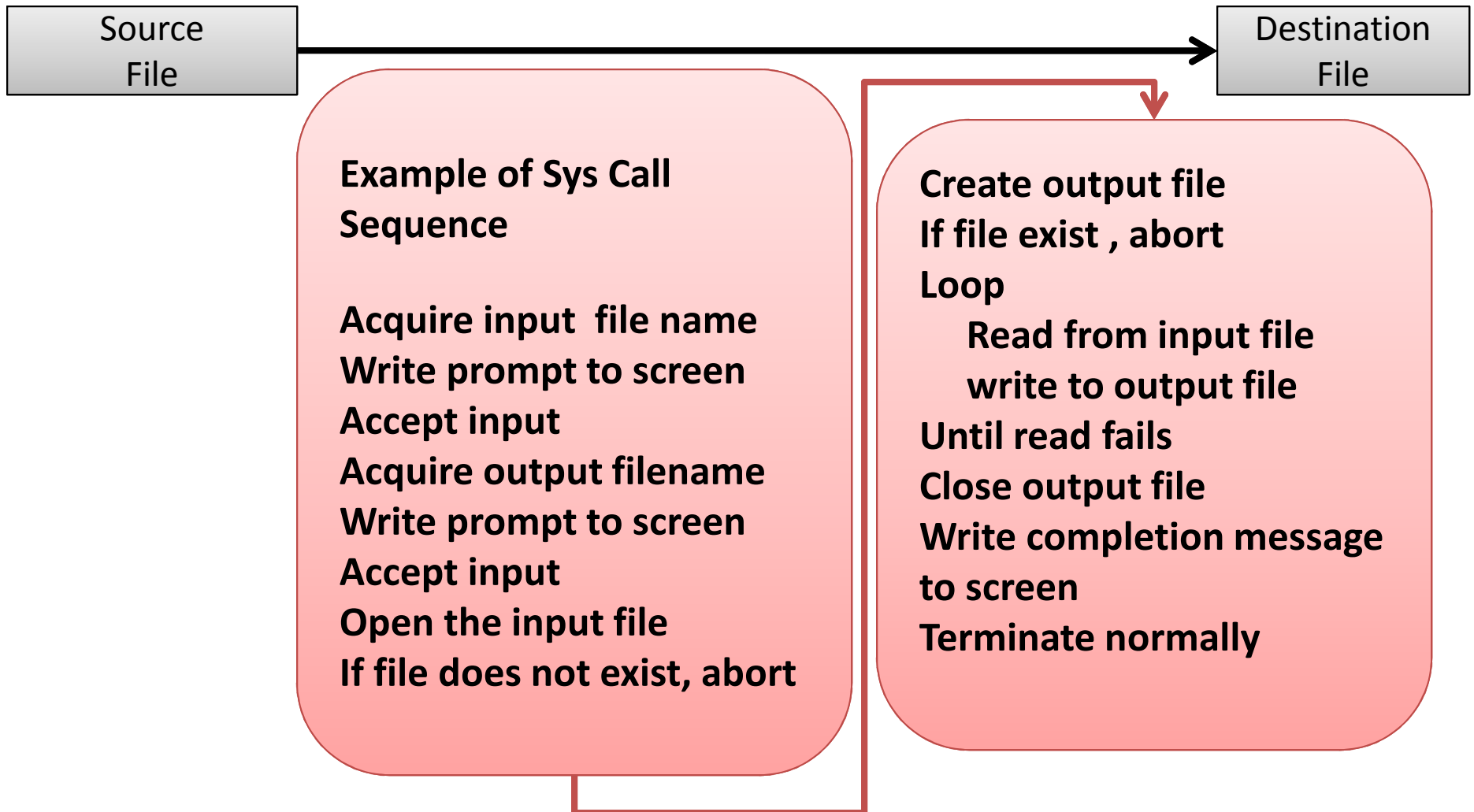
Standard C Library Example

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



Example of System Calls

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



Example of Standard API

- Standard API to read data from file or I/O

```
#include <unistd.h>
ssize_t read(int fd, void *buff, size_t count);
```

- I/O device abstracted as File

\$ man 2 read

// 2nd manual

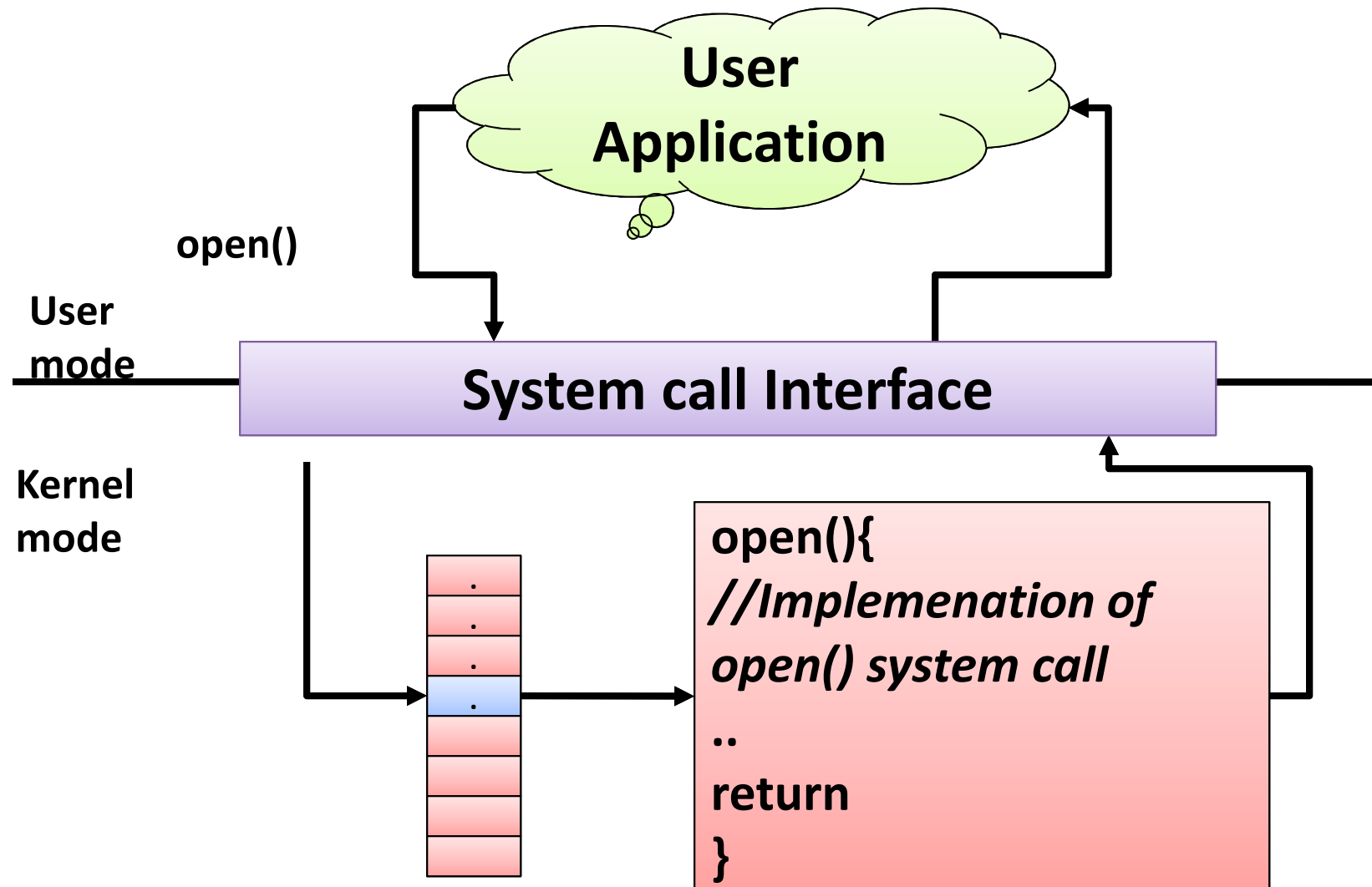
System Call Implementation

- Typically, a number associated with each system call
 - **System-call interface** maintains a table indexed according to these numbers
- The system call interface
 - invokes the intended system call in OS kernel
 - And returns status of the system call and any return values

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)

API–System Call–OS Relationship



Thanks