CSC 360: Introduction to Operating Systems

Dr. Ahmad Abdullah
abdullah@uvic.ca

Department of Computer Science
University of Victoria
Spring 2023

Lectures: ECS 123, MWR 2:30 – 3:20 pm
Office: ECS 617, MR 11:30 – 1:00 pm, W 10:00 – 11:00 am

02- Operating-System Structures

Ahmad Abdullah, PhD
abdullah@uvic.ca
https://web.uvic.ca/~abdullah/csc360

Lectures: MWR 2:30 – 3:20 pm
Location: ECS 123

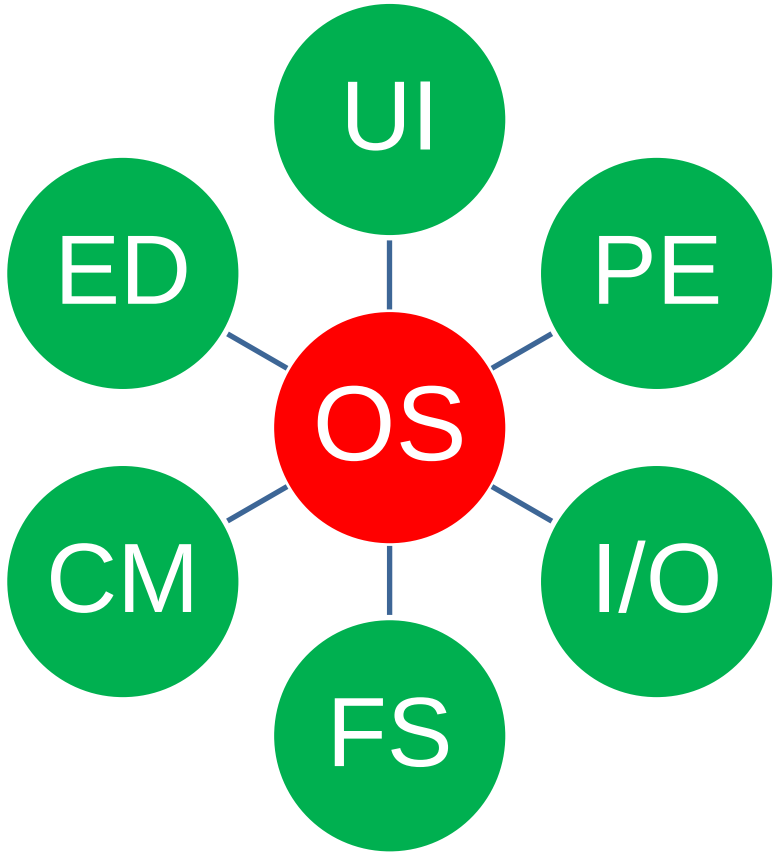
Outline

- Operating System Structures: **The System Interface**
  - Operating **System Services**
  - Briefly: OS interface presented to user
  - **System Calls**
  - **Types** of System Calls
  - **System Programs**

Covered topics from CH2 of the textbook.

OS System Services

- Operating systems are designed:
  - to provide an environment for execution ...
  - ... of **programs** and **services** ....
  - ... to **programs** and **users**.
- Set of OS services provides functions that are **directly helpful to the user**
  1. User interface
  2. Program execution
  3. I/O operations
  4. File-system manipulation
  5. Communications
  6. Error detection



OS System Services (cont.)

i. **User Interface**
   - Almost all operating systems have a user interface (UI)
   - Varies between command-line (CLI) and graphics user interface (GUI)
   - Also: **batch system** interface
   - Note: interfaces for phones and tablets now even go beyond touchscreens (e.g., Apple's Siri for voice commands)

ii. **Program execution**
   - Locating code for a program within attached storage
   - Loading this program into main memory
   - Running that program...
   - ... were execution terminates either normally or abnormally (i.e., indicating an error)

OS System Services (cont.)

iii. I/O operations

- Running programs may require data for processing ...
- ... or storage for results ...
- which may involve either a file or some other I/O device

iv. File-system operations

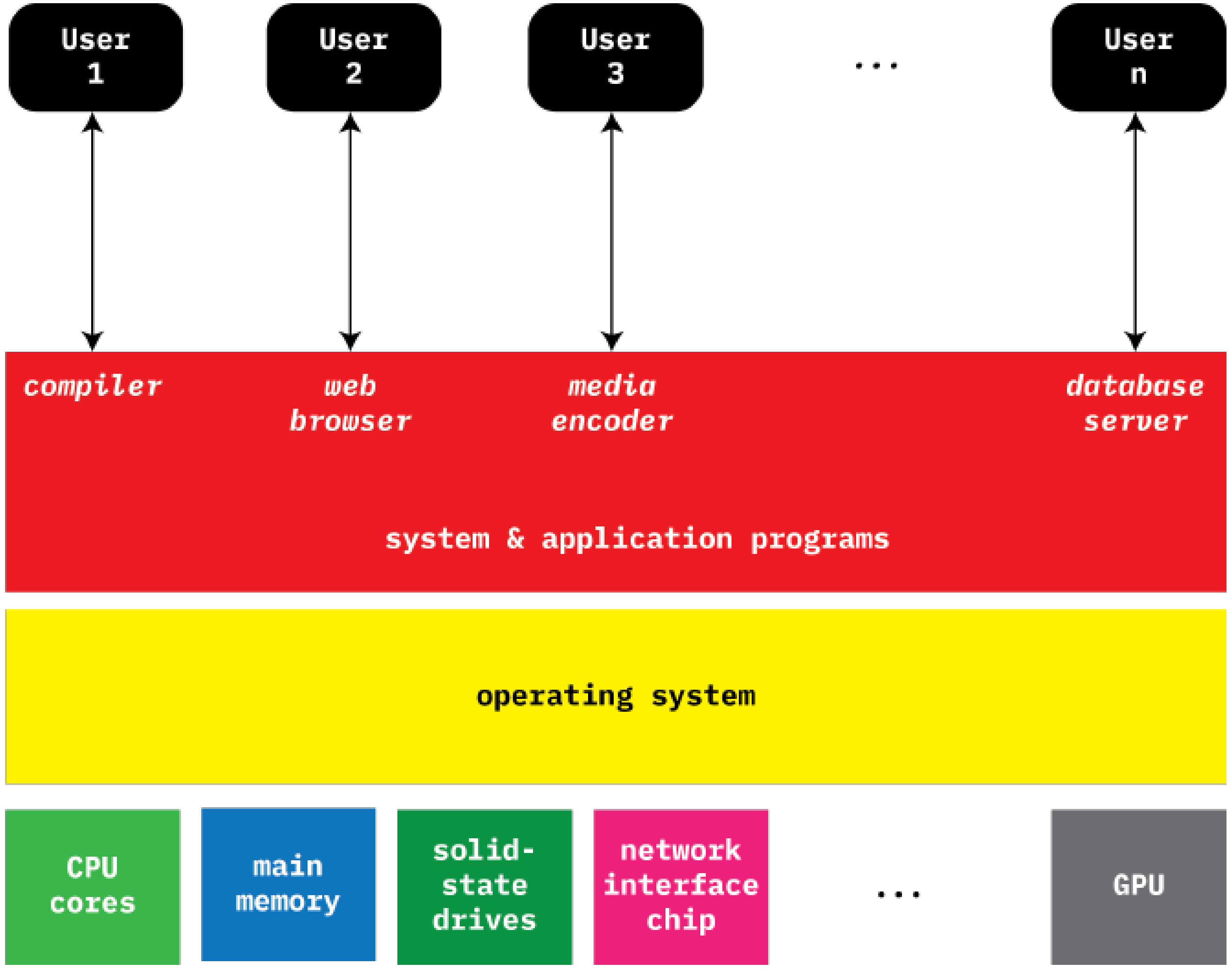
- Later in the course we will look more closely at file-system implementation
- Programs normally composed in a way that reads and writes files ...
- ... reads and writes directories ...
- ... creates and deletes files and directories ...
- ... obtain filesystem metadata ...
- ... and manage permissions.

OS System Services (cont.)

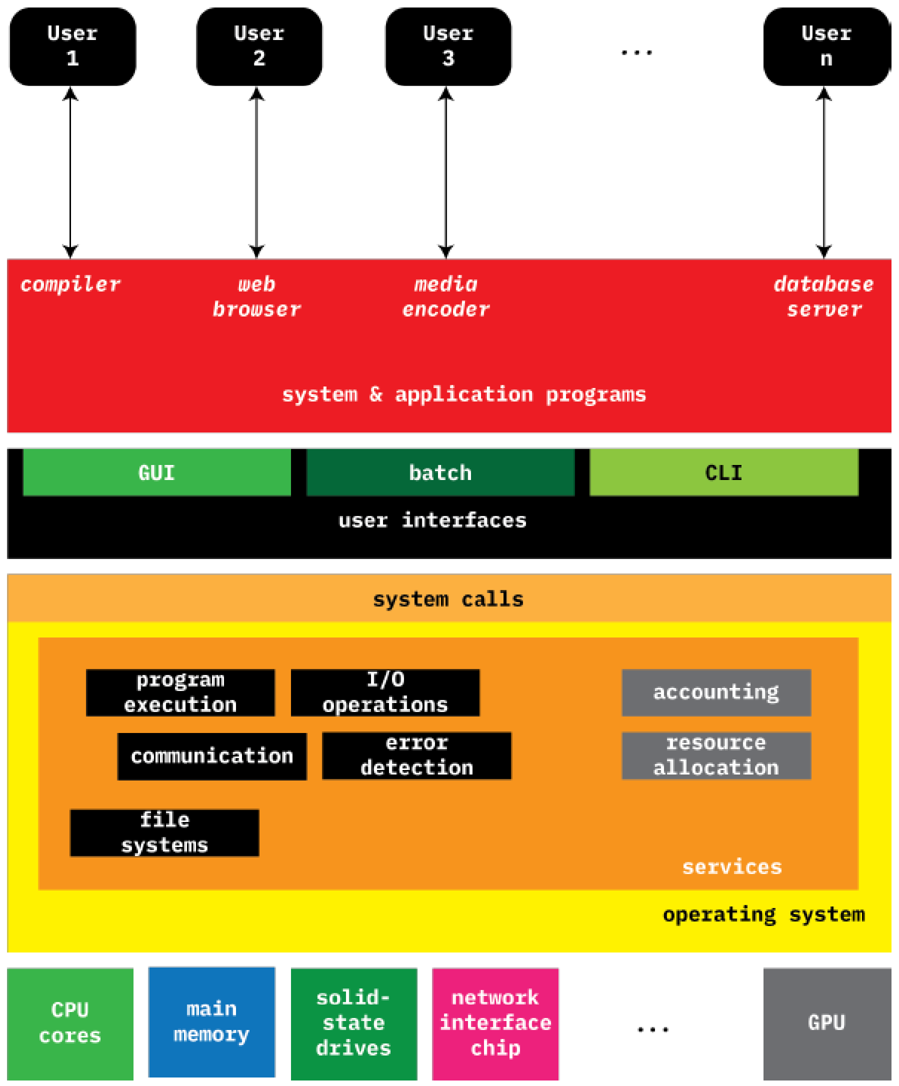
v. **Communication**
  - Processes (i.e. running programs) may exchange information / data
  - These exchanges can occur completely within the same computer
  - However, they can also occur between running programs on different computers
  - Communication mechanism may one using **shared memory** or through **message passing** (i.e., packets moved by the OS).

vi. **Error detection**
  - OS is always be monitoring for possible errors
  - These may occur in the **CPU**, in the **memory hardware**, in **I/O devices**, or in the **user program** itself
  - For each error, OS should **take appropriate steps to respond** in such a way that computer behaves **consistently and correctly**
  - OS can provide **debugging facilities** which can greatly enhance both the user's and the programmer's ability to use the OS efficiently.

OS System Services (cont.)



OS System Services (cont.)



System calls

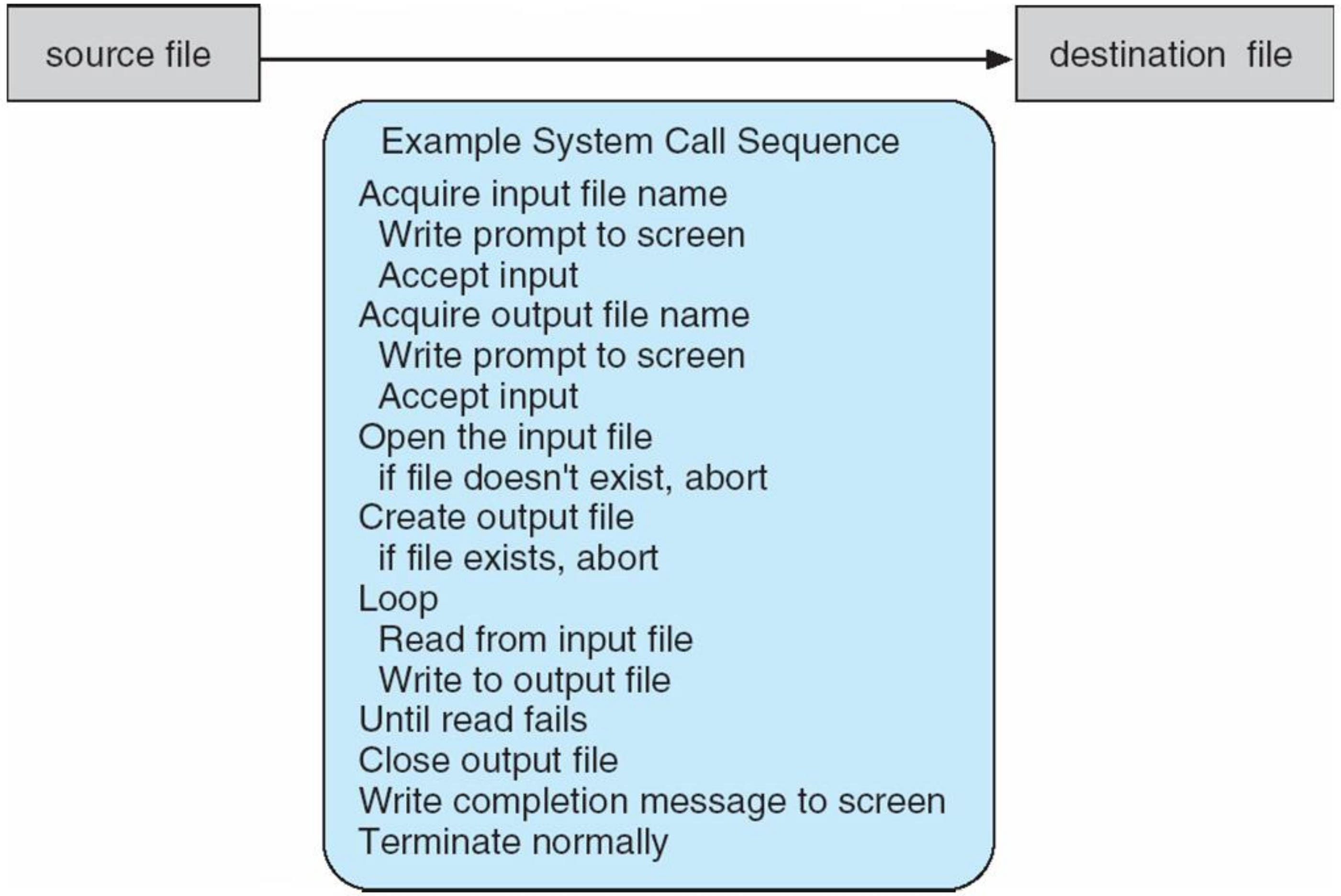
- Recall the notion of an **API**
  - Application Programming Interface
  - Provides the set of methods / functions available from a particular language or library environment
- **System calls** are the programming interface to the services as provided by the OS
  - We sometimes even refer to them as **system-call interface**
- Implementation of interface is usually in a high-level language such as C or C++
  - Oftentimes functions in this interface are called directly by methods / functions in a related API rather than directly by the programmer

System calls (cont.)

- Two most-widely used system-oriented APIs at present
  - **Win32 API** (note that even though many installations are currently 64 bit, we do not yet refer to them as Win64 API)
  - **POSIX API**: Standardized Unix system-call interface (Portable Operating System Interface, also known as ISO/IEC 9945)
- Other programming APIs are built on top of these
  - Pretty much all versions of Unix (including macOS) use POSIX
  - Java API sits on top of either Win32 or POSIX
- Although we will focus on Unix/POSIX in this course...
  - ... the names used for system call instances will be more generalized.

Example of System Calls

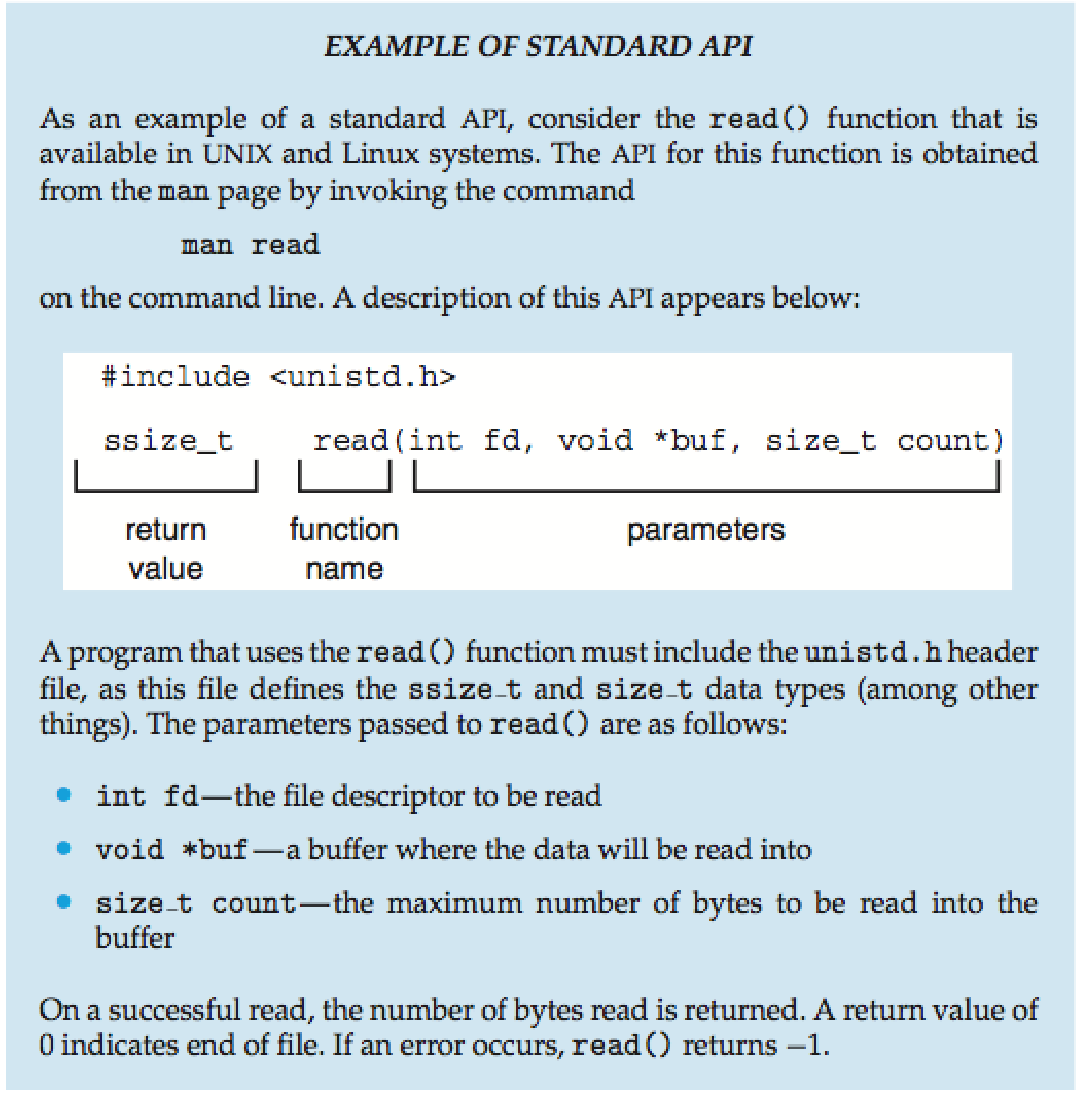
- System call sequence to copy the contents of one file to another file



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 System Calls (cont.)



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>
```

$$\underbrace{\texttt{ssize\_t}}_{\text{return value}}\quad \underbrace{\texttt{read}}_{\text{function name}}\underbrace{\texttt{(int fd, void *buf, size\_t count)}}_{\text{parameters}}$$

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:

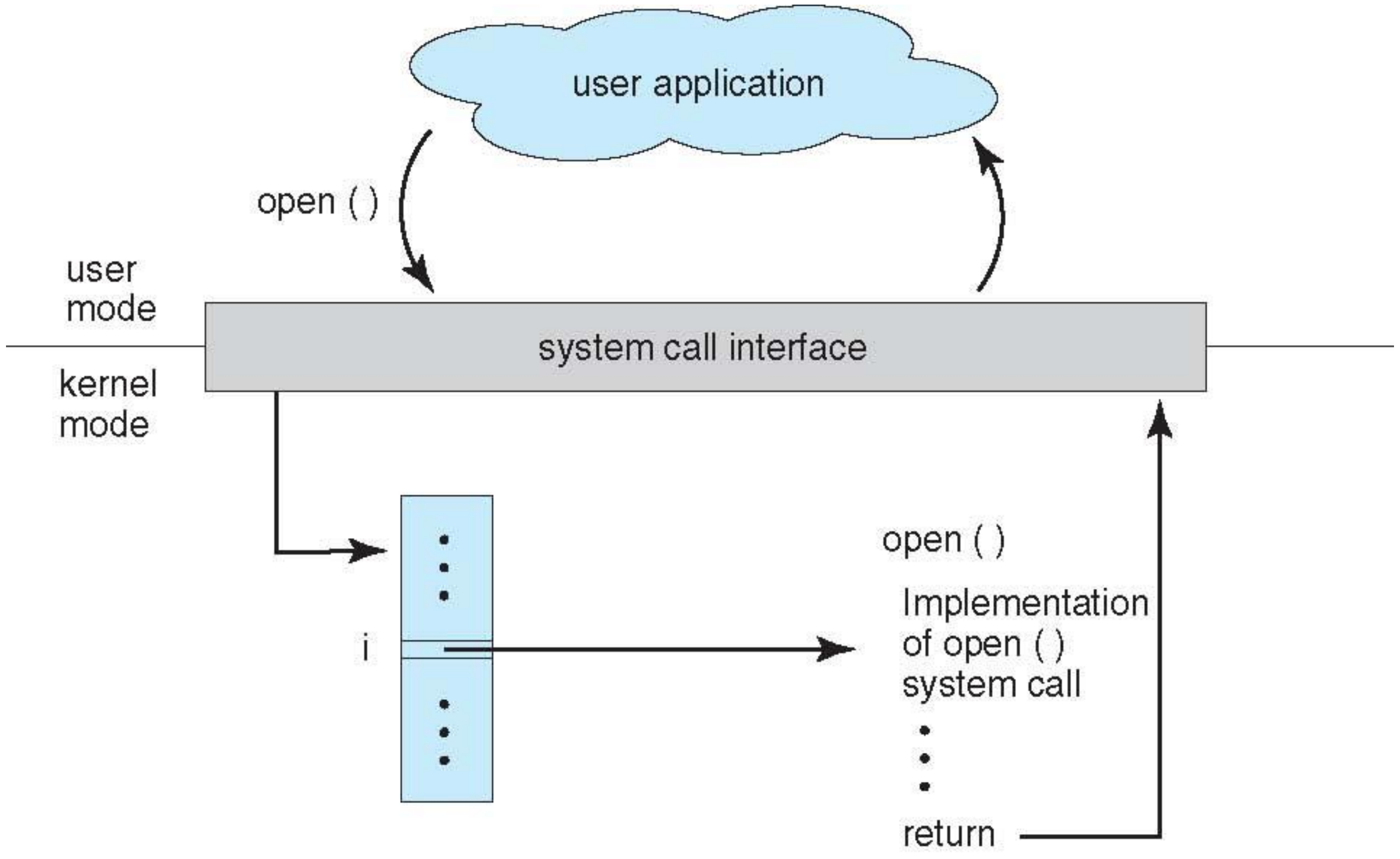
- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

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

- Typically each individual system call operation boils down to a single **number**
  - The actual system-call interface is maintained via a table indexed according to these numbers.
  - **The code for the OS kernel uses these numbers as indexes into a large switch statement in the kernel**
- That is:
  - The system-call interface invokes the intended system call function itself in the OS kernel by providing the number...
  - ... and the OS kernel returns the status of the system call and any return values.

API – System Call – OS Relationship



Any Questions?