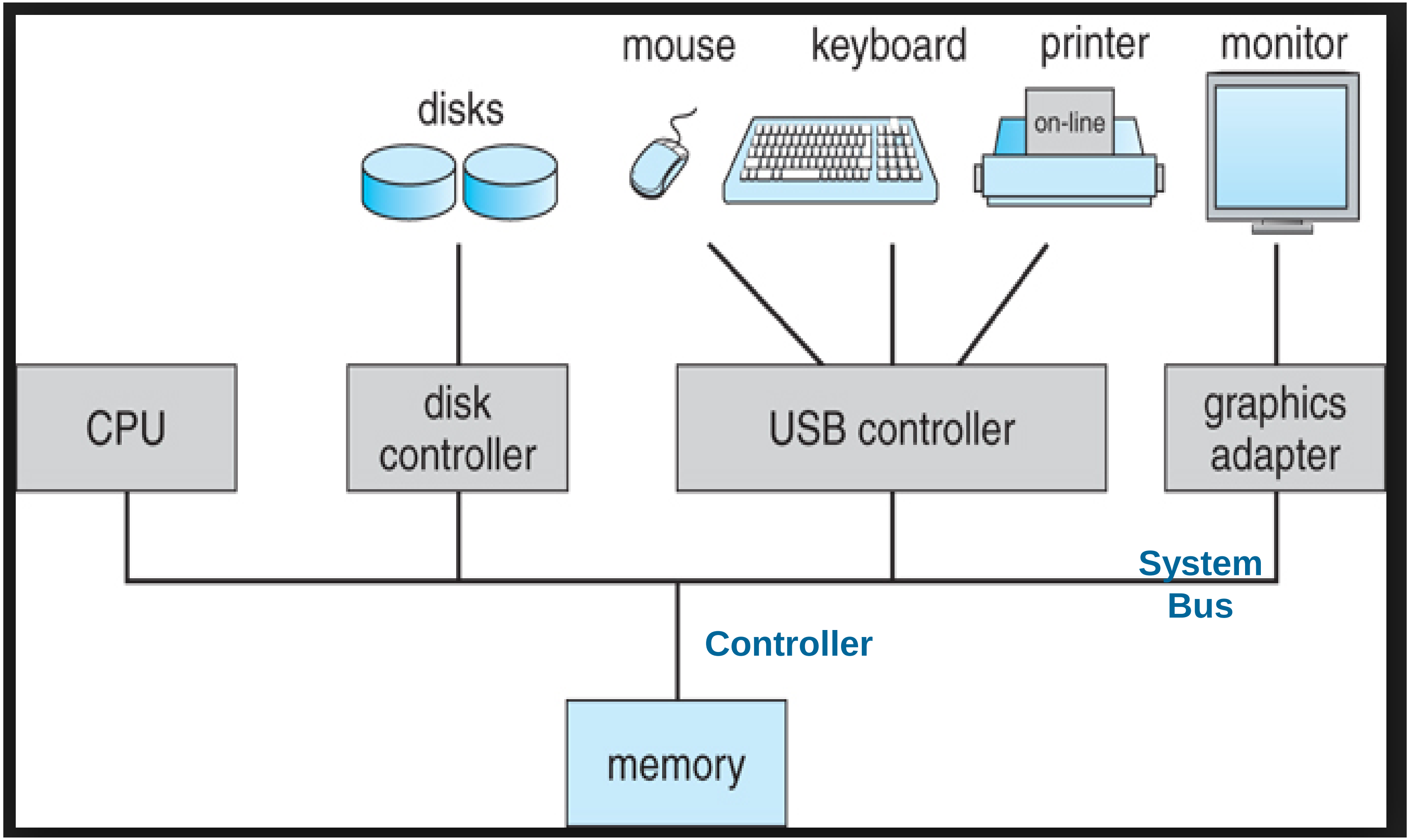
Chapter : Computer System Structure and Interrupts

(Computer System Structure and Interrupts)

By: Navjot Kaur

Computer System



System Bus

Controller

- When system is powered on, kernel is first loaded into the memory.
- Kernel execute a init program, and waits for some event (response) to occur, This event is known as Interrupt.
- OS initializes all the aspects of system, from CPU Registers → Device Controller → Memory.
- The occurrence of any event is signaled by interrupt.

Type of Interrupts

- Interrupts can be :
  - Generated by Hardware (Hardware Interrupt)
  - Generated by Software (Software Interrupt)

Hardware Interrupt

1. Hardware interrupts are used by **devices** to communicate that they **require attention from the operating system**.
2. Hardware interrupts by sending signal to CPU **via system bus.**
3. Hardware interrupts are referenced by an ***interrupt number*****.**
4. These numbers are mapped with hardware that created the interrupt. This enables the system to monitor/understand **which device created the interrupt and when it occurred**.

Software Interrupt/ Trap

- Interrupt generated by executing a instruction.
- Software interrupts by a special operation called a System Call or Monitor Call.

Exp: 1. cout in C++ is a kind of interrupt because it would make a system to print something.

2. division by zero

- OS is interrupt driven, it is idle when no inputs are there to process.
- When CPU is interrupted it stops doing everything and transfers the control to a **fixed location.**
- **Fixed Location** contains the address where service for interrupt is located.
- CPU resumes after executing interrupt.

General Mechanism

- When interrupt occurs:

Interrupt transfers control to Interrupt Service Routine

Interrupt Service Routine examines the interrupt info. and calls interrupt handler.

General Mechanism

- If no. of interrupts are predefined:

Table of pointers is maintained that point to interrupt routine.

Interrupt vector is a pointer having unique id and interrupt request for interrupted devices.

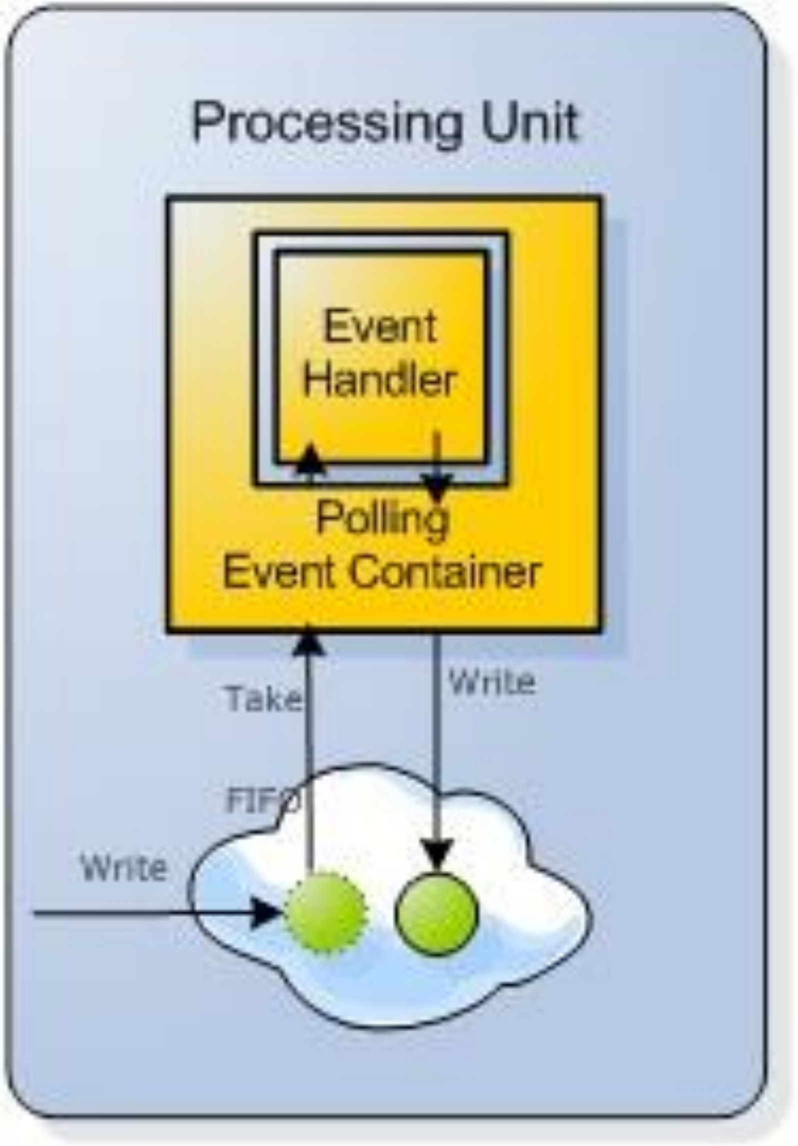
Basic steps when interrupt occurs:

1. Interrupt Occurred?
2. H/w Transfers control to OS
3. OS preserves current state of process by using Registers and Program counter
4. Determine which kind of interrupt has occurred and Provides resources
5. When interrupt is executed, address is loaded to program counter and interrupted services are resumed.

Polling

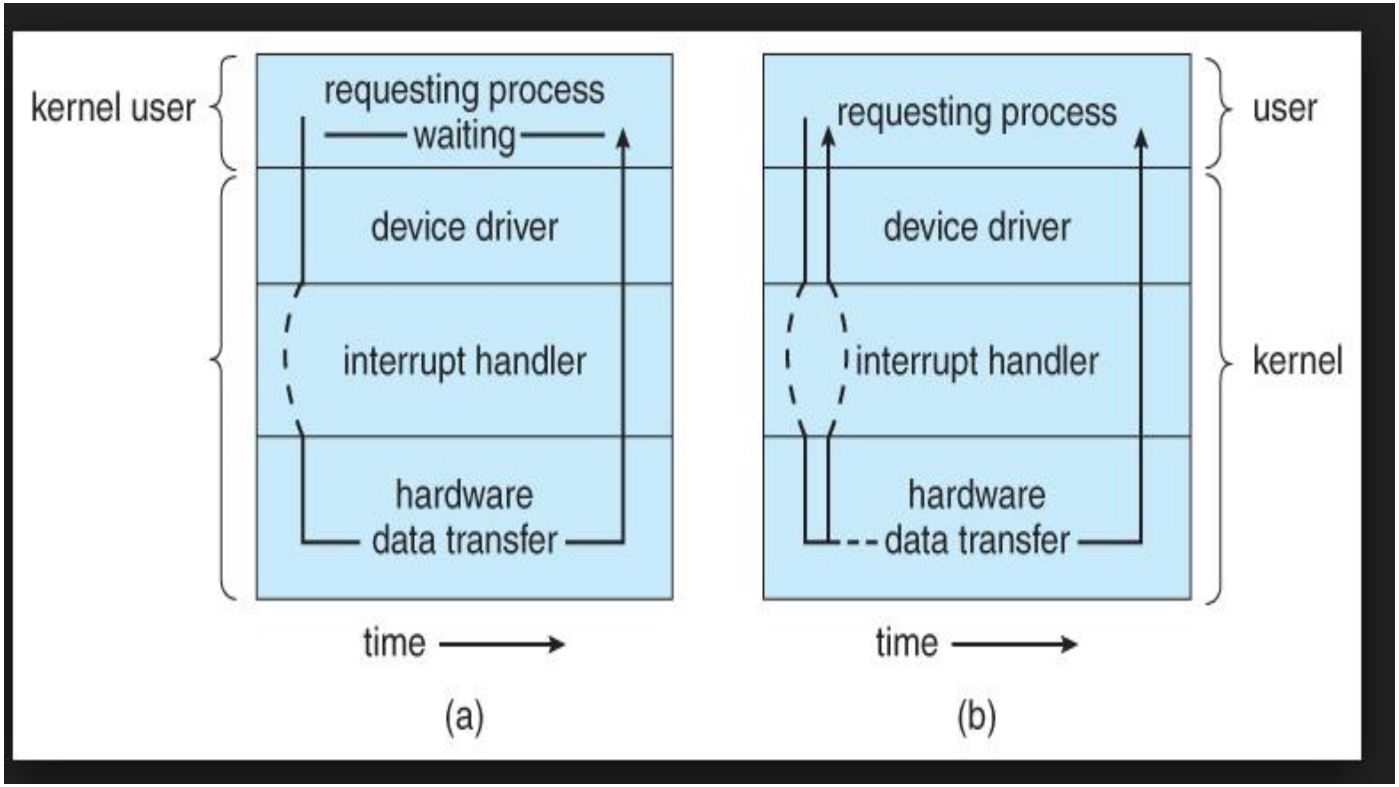
- Process where the controlling device waits for an external device to check for its readiness or state.

- Queuing all the I/O devices to see which I/O requested for the service?



I/O Interrupt

- Once the I/O is started, 2 kind of interrupts may arise:
- **Synchronous interrupt (I/O):** The control is transferred to the user process when I/O completes.
- **Asynchronous interrupt (I/O):** Returns control to user process without waiting for I/O to complete.



(a) Synchronous (b) Asynchronous

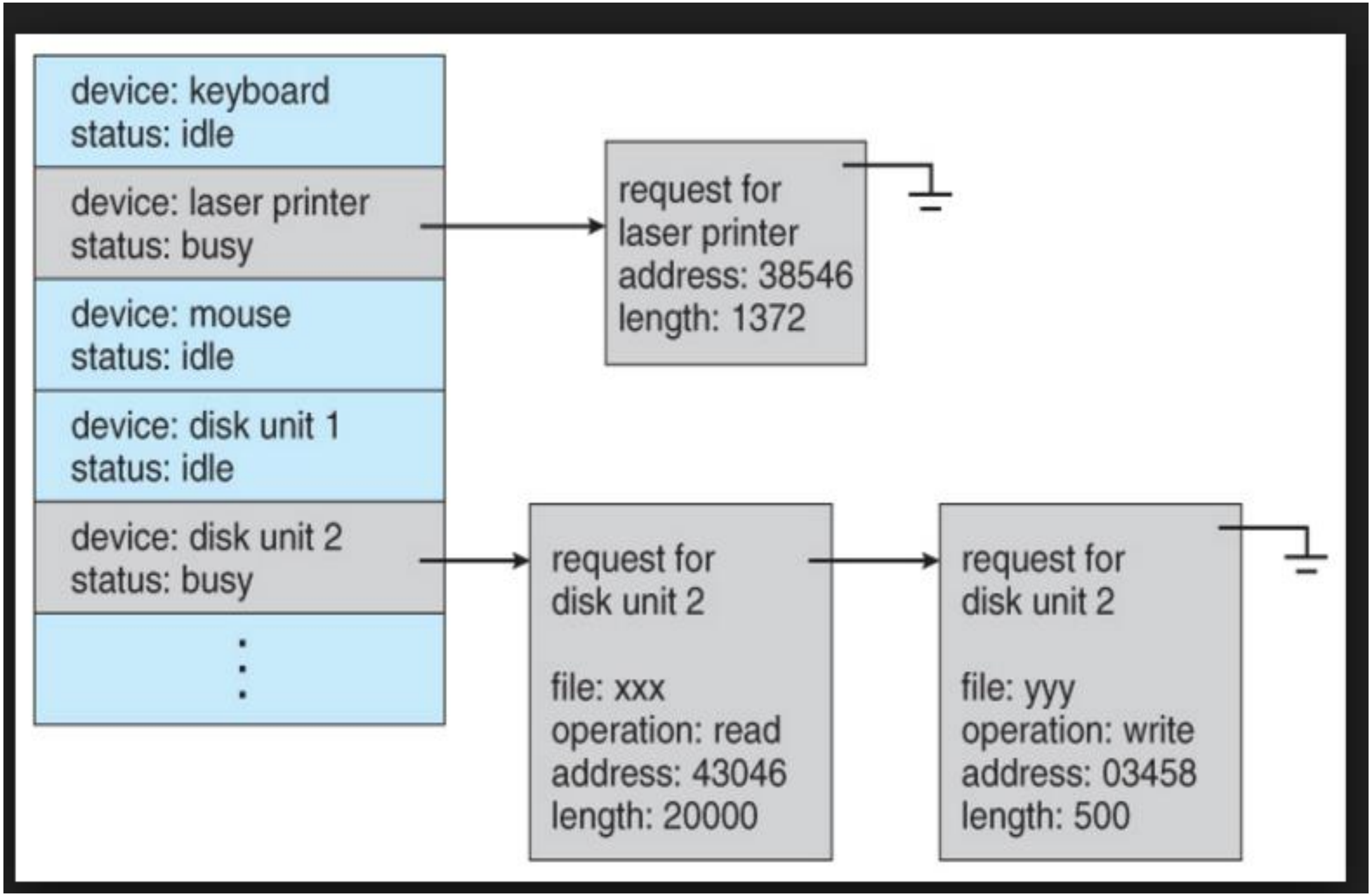
Waiting of I/O may be accomplished in 2 ways:

- Wait: it let the CPU wait till the next interrupt.

- Loop: Loop continues until interrupt is active.

Device Status Table

- To keep the track of how many I/O requests are pending
- Device Status Table is maintained.
  - Device type
  - Address
  - State (idle,busy)



device: keyboard
status: idle

device: laser printer
status: busy

device: mouse
status: idle

device: disk unit 1
status: idle

device: disk unit 2
status: busy

⋮

request for
laser printer
address: 38546
length: 1372

request for
disk unit 2

file: xxx
operation: read
address: 43046
length: 20000

request for
disk unit 2

file: yyy
operation: write
address: 03458
length: 500

Who controls the execution of programs to prevent errors and improper use of computer?

a) Resource allocator

b) Control Program

c) Hardware

d) None of the above

The operating system switches from user mode to kernel mode so the mode bit will change from?

a) 0 to 1

b) 1 to 0

c) Remain constant

d) None

Q. In which type of operating system users do not interact directly with the computer system?

a) Multiprogramming operating systems

b) Multiprocessing operating systems

c) Batch operating systems

d) Distributed operating systems

Q. What is the objective of multiprogramming operating systems?

a) Maximize CPU utilization

b) Switch the CPU among processes

c) Achieve multitasking

d) None of the above

what signal for the occurrence of an event either from the hardware or the software?

a) Bootstrap program

b) Interrupt

c) Disk Controller

d) CPU

- In which type of I/O interrupts the control return to the user program after the completion of I/O operation?

a) Synchronous I/O interrupts

b) Asynchronous I/O interrupts

c) System Call

d) Hardware

□ The device-status table contains

a) each I/O device type

b) each I/O device state

c) each I/O device address

d) all of the above

System Calls

- Allow user-level processes to request services of the operating system.

- It provides a way in which program talks to the operating system.

Why system calls are required?

- **It is a request to the operating system to perform some activity**.
- It is a **call to the kernel** in order to **execute a specific function** that controls a device or executes a instruction.

- A system call looks like a procedure call

Accessing System Calls

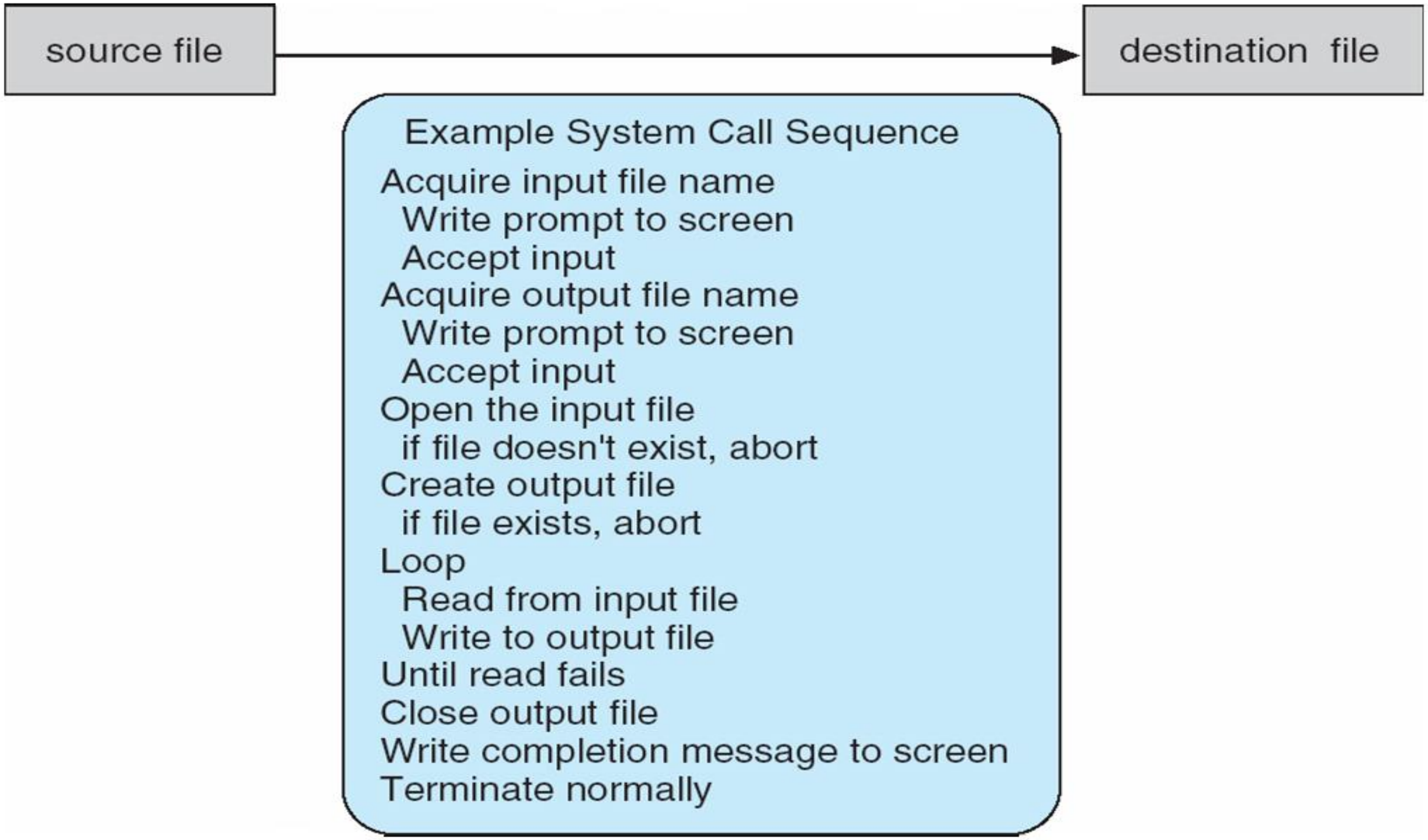
- System calls are accessed by programs via a high-level **Application Program Interface (API)** (provides run time environment)

- System calls are implemented via API, API’s interact with kernel of OS.

- 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)

Example of System Calls

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



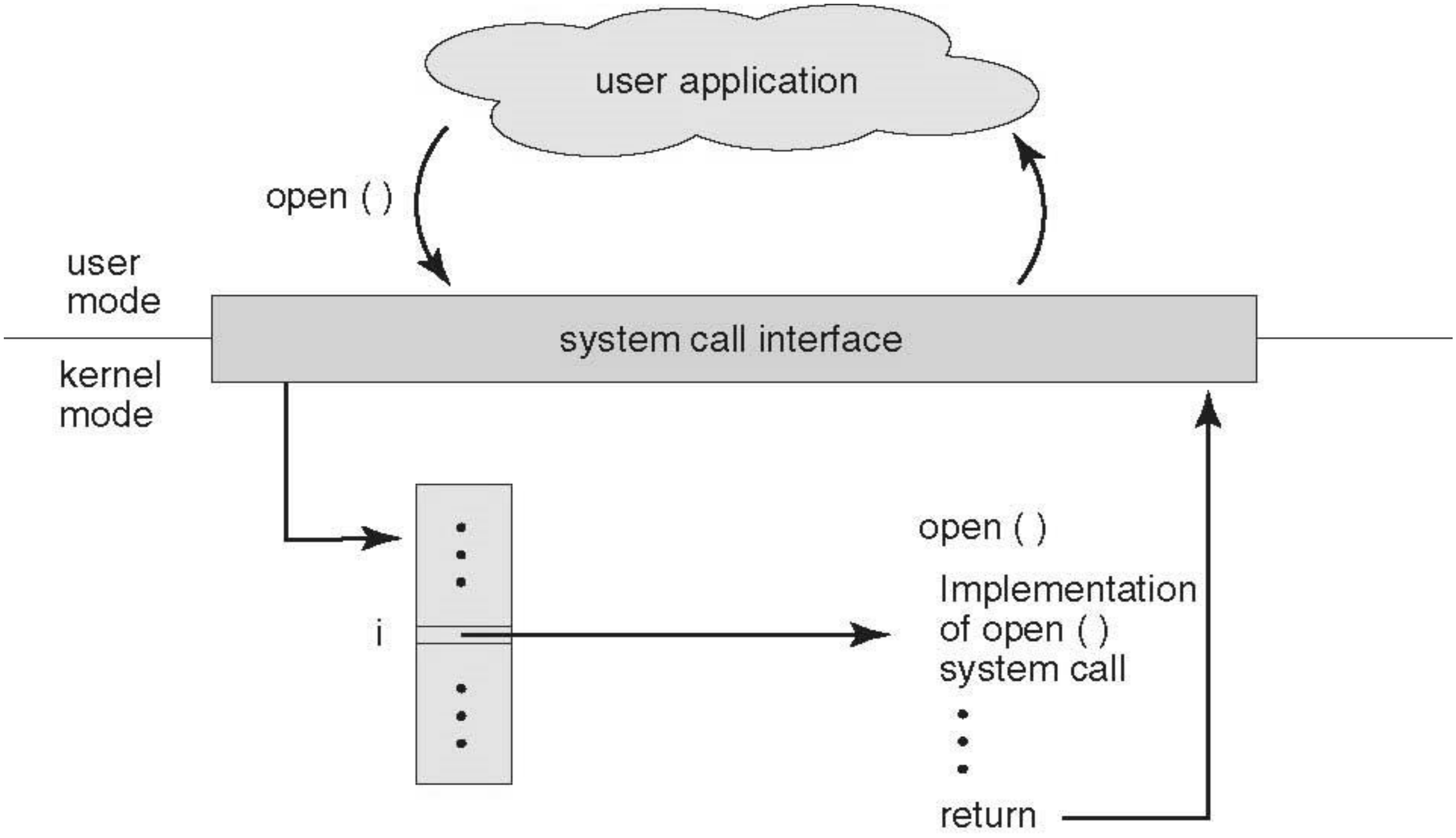
Example System Call Sequence

```
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
```

System Call Implementation

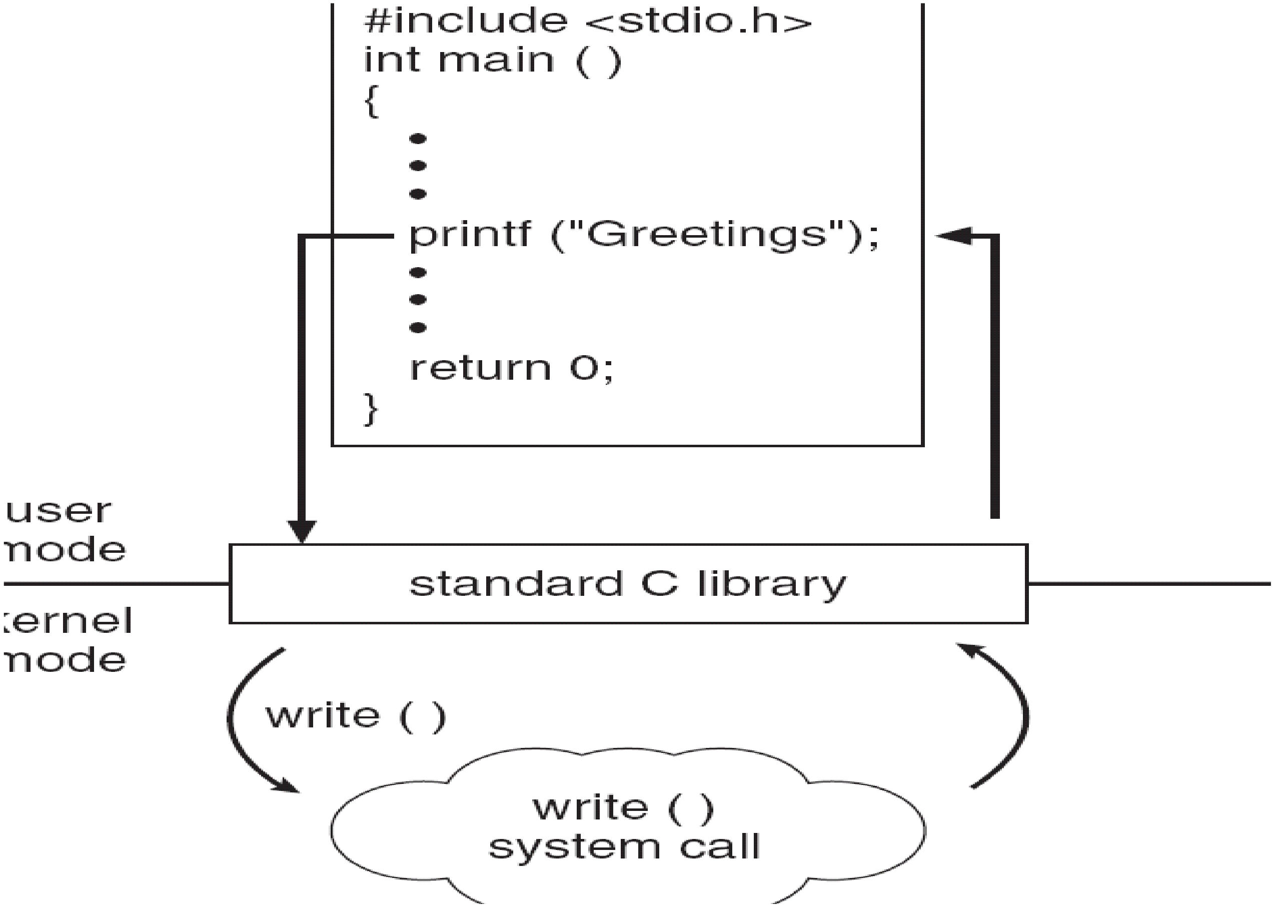
- A number is associated with each system call
  - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call with a return value.
- The caller needs to know nothing about how the 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

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 Call Dispatch

1. Kernel assigns system call number to each system number
2. Kernel initializes system call table, mapping system call number to functions implementing the system call
3. User process sets up system call number and arguments
4. User process runs

System Call Dispatch

5. Hardware switches to kernel mode and calls kernel’s interrupt handler for user process (interrupt dispatch)

6. Kernel looks up syscall table using system call number

7. Kernel calls/invokes the corresponding function

8. Kernel returns by running interrupt return and processing the actual program.

System Call Parameter Passing

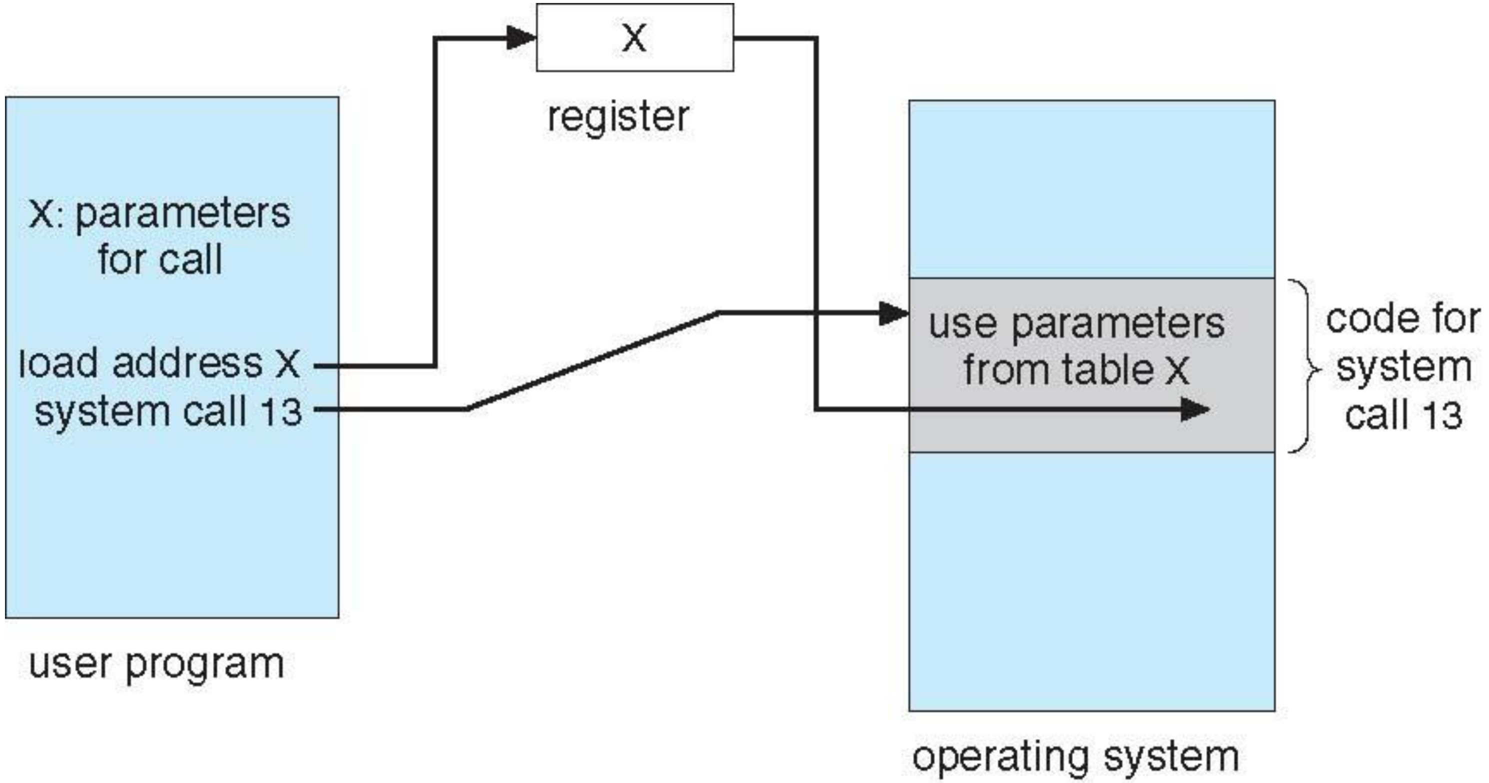
Passing Parameters to System Calls:

- Information required for a system call vary according to OS and call.

- **Three general methods used to pass parameters to the OS**

1. **Pass the parameters in *registers***
   - When parameters are < 6.
2. **Parameters stored in a *block,* or table**, in memory, and address of block passed as a parameter in a register. (6 or more)
   - This approach taken by Linux and Solaris
3. **Parameters placed**, or *pushed,* **onto the *stack*** by the program and *popped* off the stack by the operating system.

Parameter Passing via Table



Types of System Calls

5 Categories

- Process Control
  - end, abort
  - load, execute
  - create process, terminate process
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory

- File Management
  - create file, delete file
  - open, close file
  - read, write, reposition
  - get and set file attributes

Types of System Calls (Cont.)

- Device Management
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices
- Information Maintenance
  - get time or date, set time or date
  - get system data, set system data
  - get and set process, file, or device attributes

- Communications
  - create, delete communication connection
  - send, receive messages
  - transfer status information
  - attach and detach remote devices

Process Control Calls

- fork() – create a new process

  `pid = fork();`

  - The *fork*() function shall create a new process. The new process (child process) shall be an exact copy of the calling process (parent process) except some process' system properties
  - It returns 'twice'
    - return value == 0 ... child
    - return value > 0 ... parent (returned value is the child's *pid*)

- exit() – terminate a process

  `void exit(int status);`

  - The *exit*() function shall then flush all open files with unwritten buffered data and close all open files. Finally, the process shall be terminated and system resources owned by the process shall be freed
  - The value of 'status' shall be available to a waiting parent process
  - The *exit*() function should never return

File Access Calls

- open – open file

```
fd = open(const char *path, int oflag, ...);
```

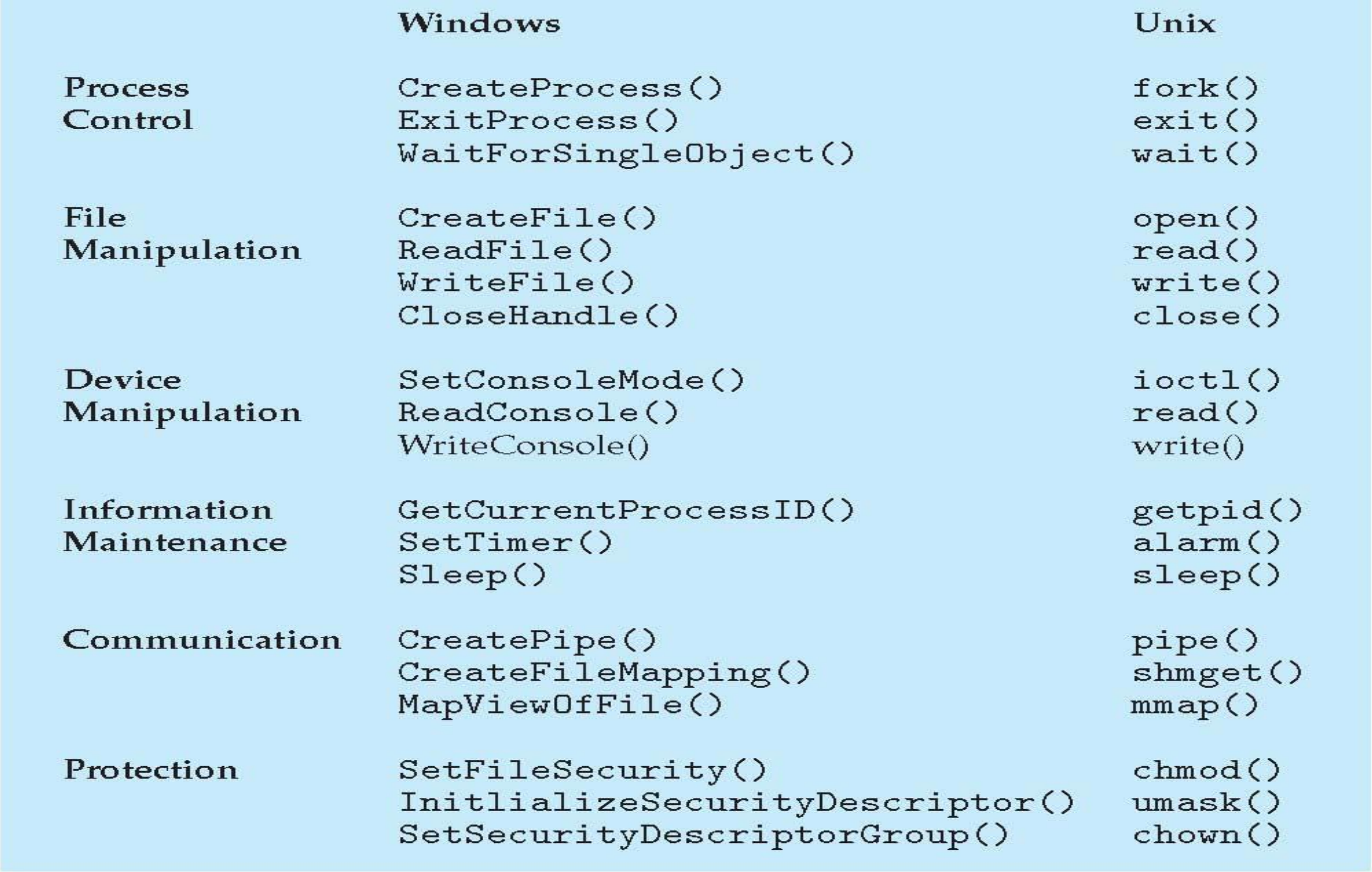
  - The *open()* function shall establish the connection between a file and a file descriptor. The file descriptor is used by other I/O functions to refer to that file. The `path` argument points to a pathname naming the file.
  - The parameter `oflag` specifies the open mode:
    - ReadOnly, WriteOnly, ReadWrite
    - Create, Append, Exclusive, ...

- close – close a file descriptor

```
err = close(int fd);
```

  - The *close*() function shall deallocate the file descriptor indicated by *fd*. To deallocate means to make the file descriptor available for return by subsequent calls to *open*() or other functions that allocate file descriptors.
  - When all file descriptors associated with an open file description have been closed, the open file description shall be freed.

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()

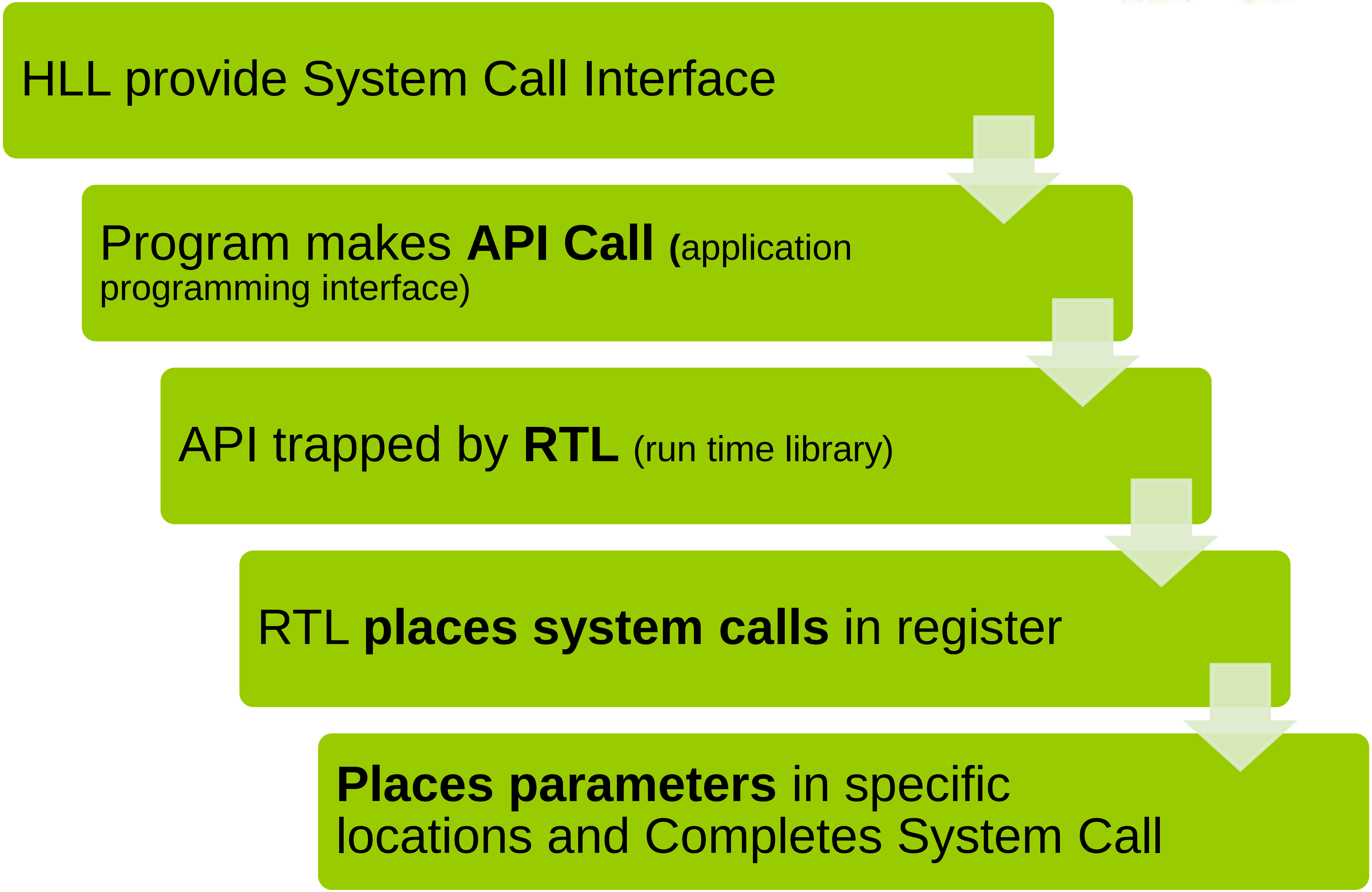
Device Manipulation	SetConsoleMode()
	ReadConsole()
	WriteConsole()
		ioctl()
		read()
		write()

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



HLL provide System Call Interface

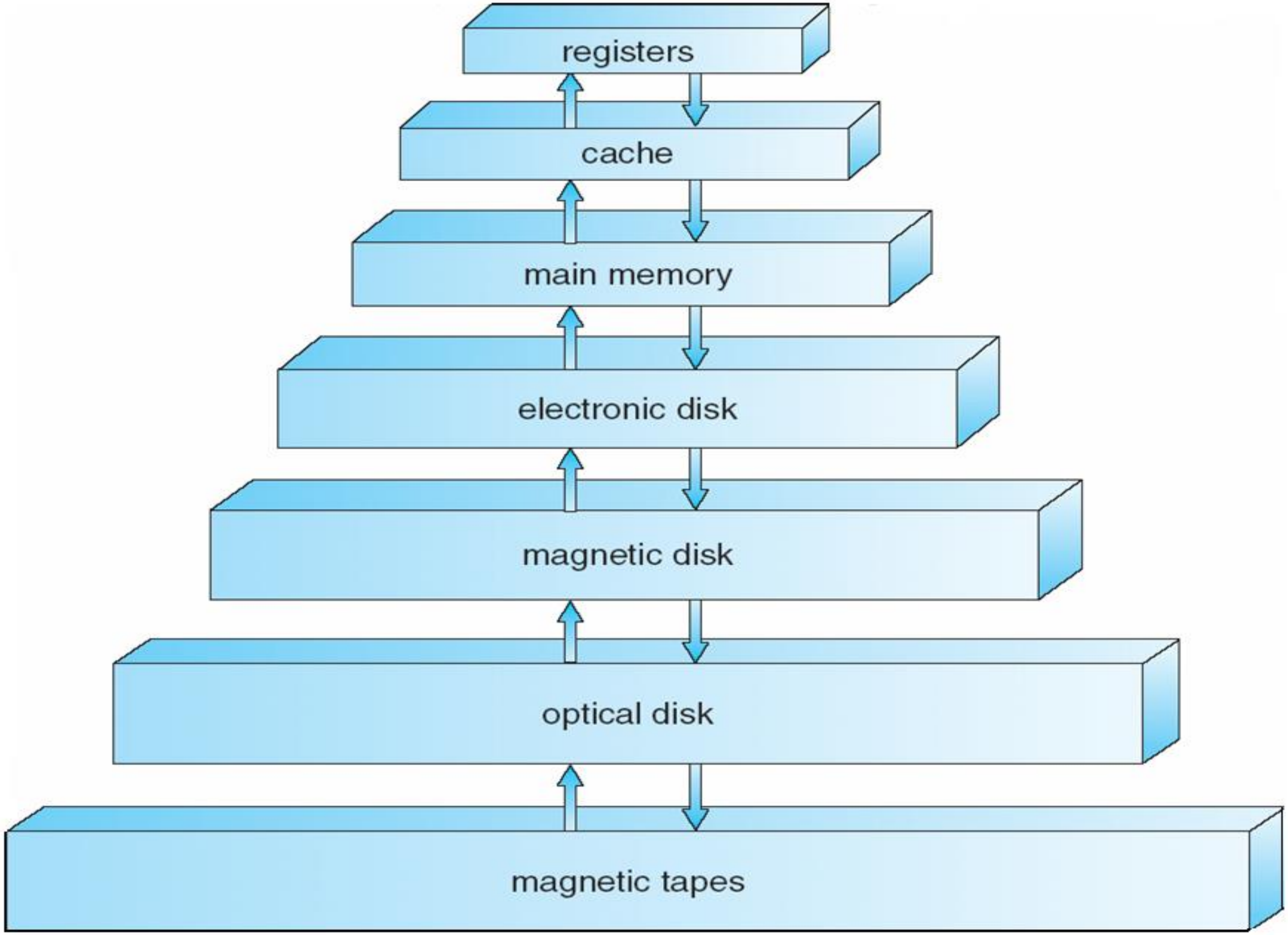
Program makes **API Call** (application programming interface)

API trapped by **RTL** (run time library)

RTL **places system calls** in register

Places parameters in specific locations and Completes System Call

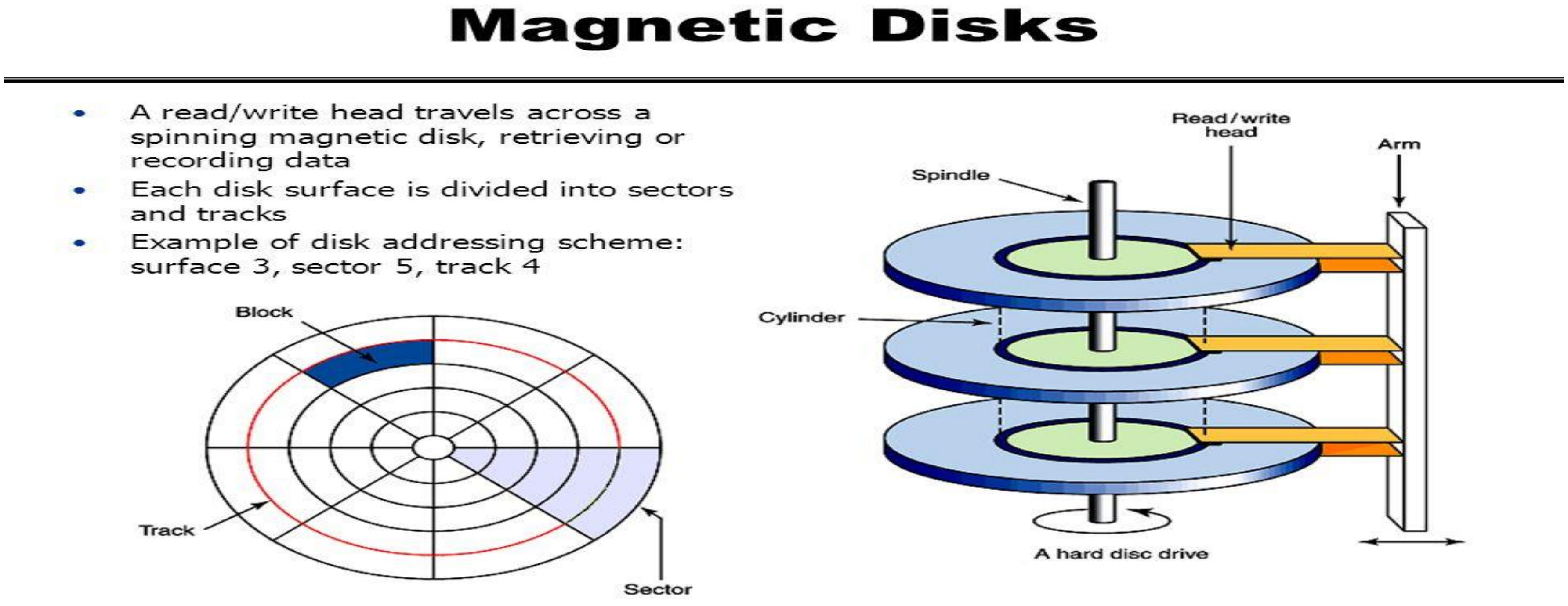
Storage Structure and Hierarchy



Magnetic Tape

Optical Tape

Main Memory (RAM)



Magnetic Disks

- A read/write head travels across a spinning magnetic disk, retrieving or recording data
- Each disk surface is divided into sectors and tracks
- Example of disk addressing scheme: surface 3, sector 5, track 4

Any Query ???

Thank You