COS3023 OPERATING SYSTEMS AND CONCURRENCY

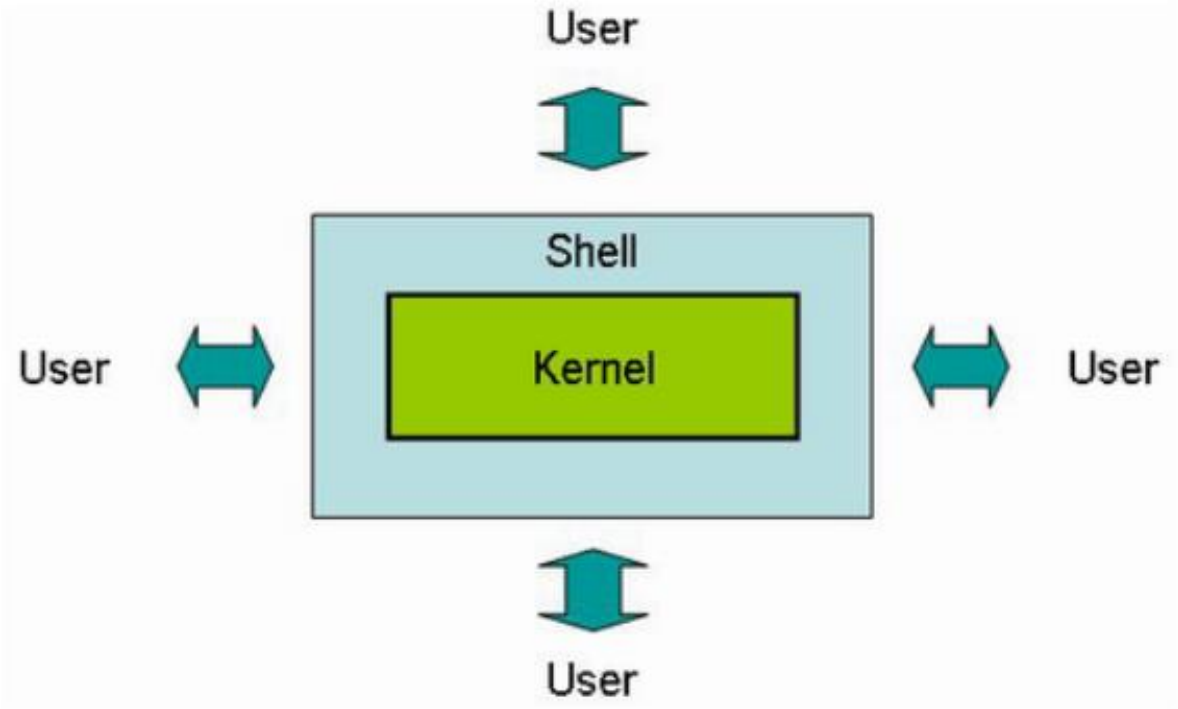
Topic 1- Operating System – Introduction (Part 3)

Lecturer : Ms Shafrah

PRESENTATION OUTLINE

- Kernel
- System Calls

KERNEL

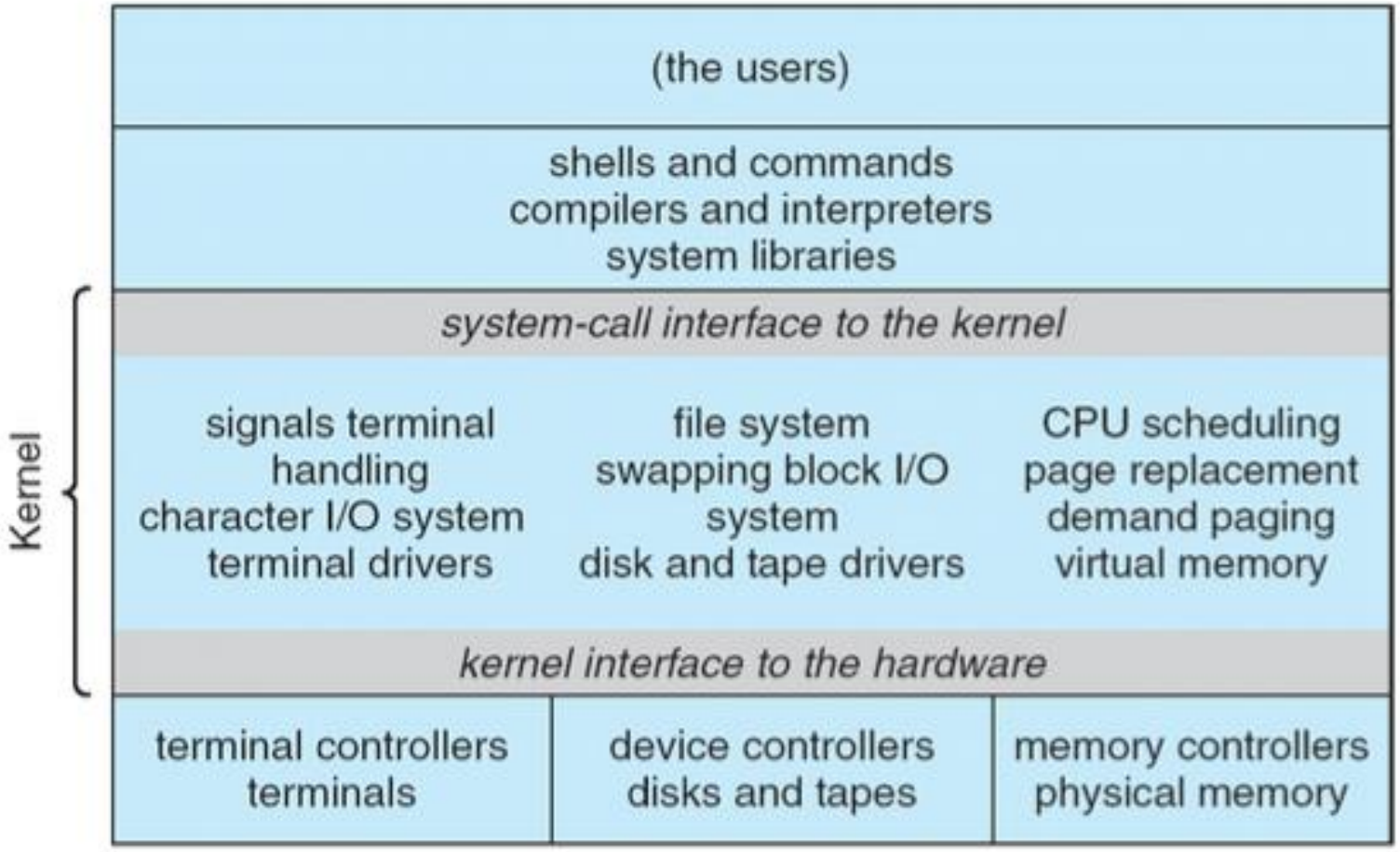


The shell as the interface between the kernel and the user.

KERNEL

- **Kernel** is the most important component of an OS, responsible for communication between hardware and software.
- Its purpose is to translate commands that can be understood by the computer.
- When a process makes a request to the kernel, it is called System Call.
- A kernel is provided with a protected space, which is a separate area of memory that are not accessible by other application programs.
- Most popular kernel types are monolithic, microkernel and hybrid.

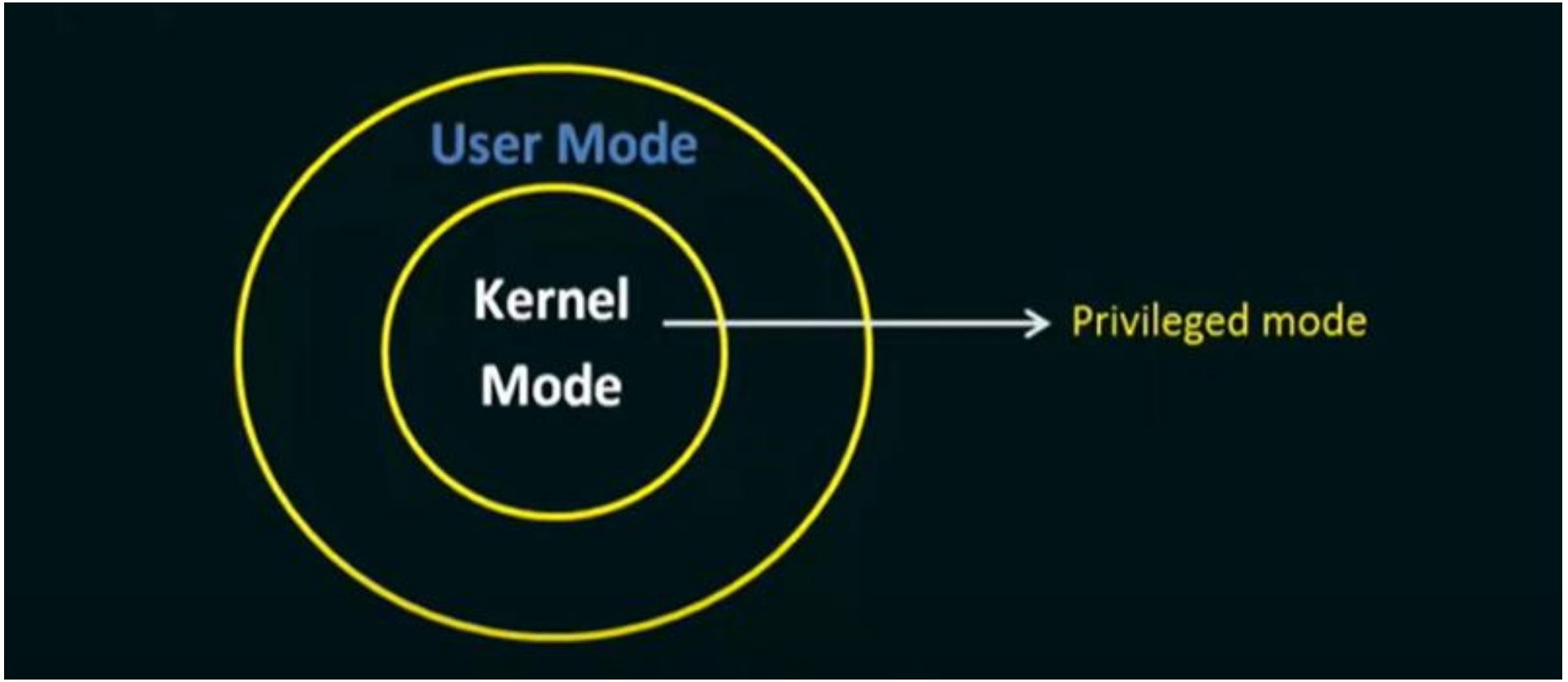
EXAMPLE OF UNIX SYSTEM STRUCTURE



KERNEL

- To protect the system from aberrant users and processors, some instructions are restricted to use only by the OS.
- The protection in kernel are divided into two; **kernel mode** and **user mode**.
- In user mode, users may not
  address I/O directly
  use instructions that manipulate the state of memory
  set the mode bits that determine user or kernel mode
  disable and enable interrupts
  halt the machine
- In kernel mode, the OS can do all these things.

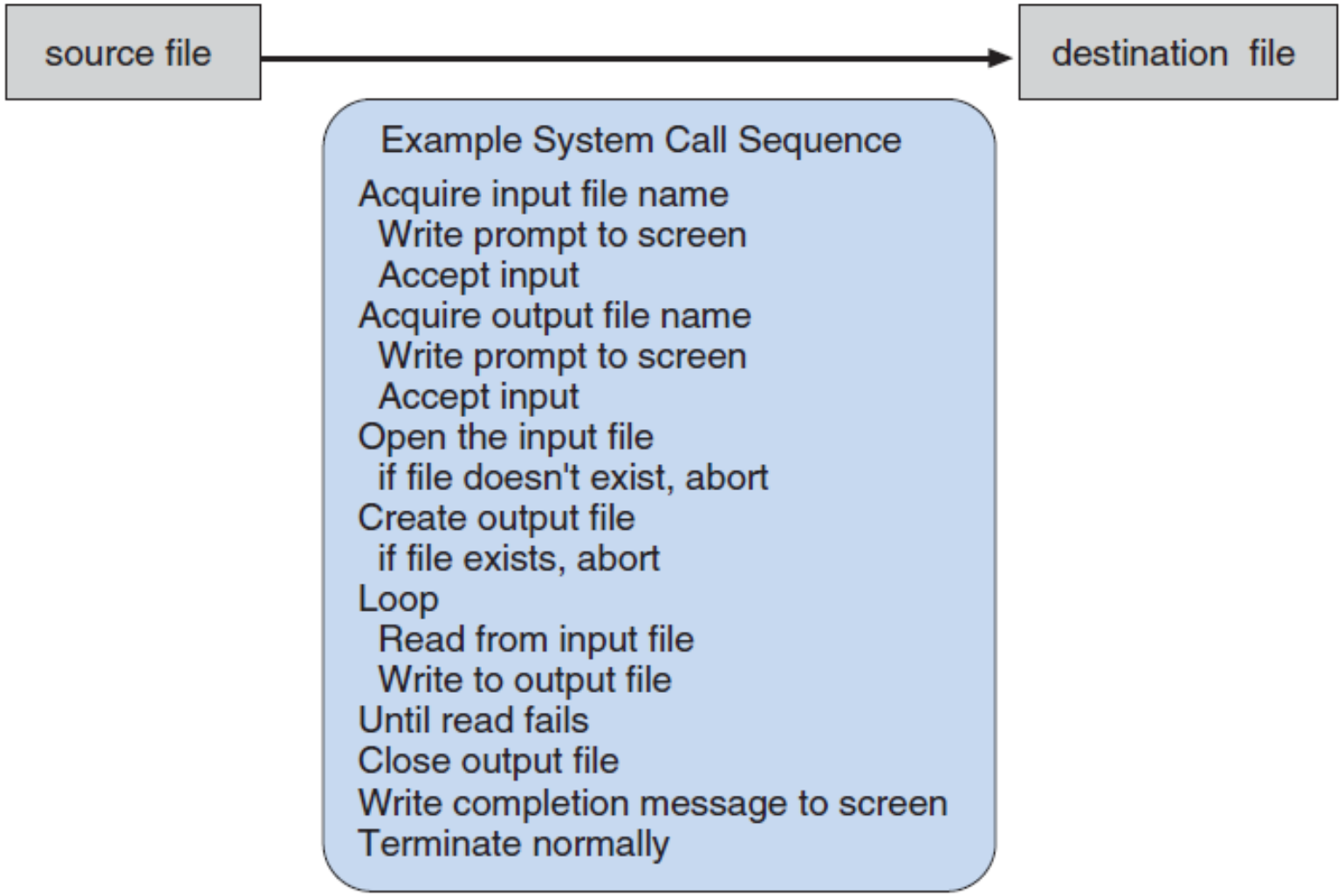
KERNEL



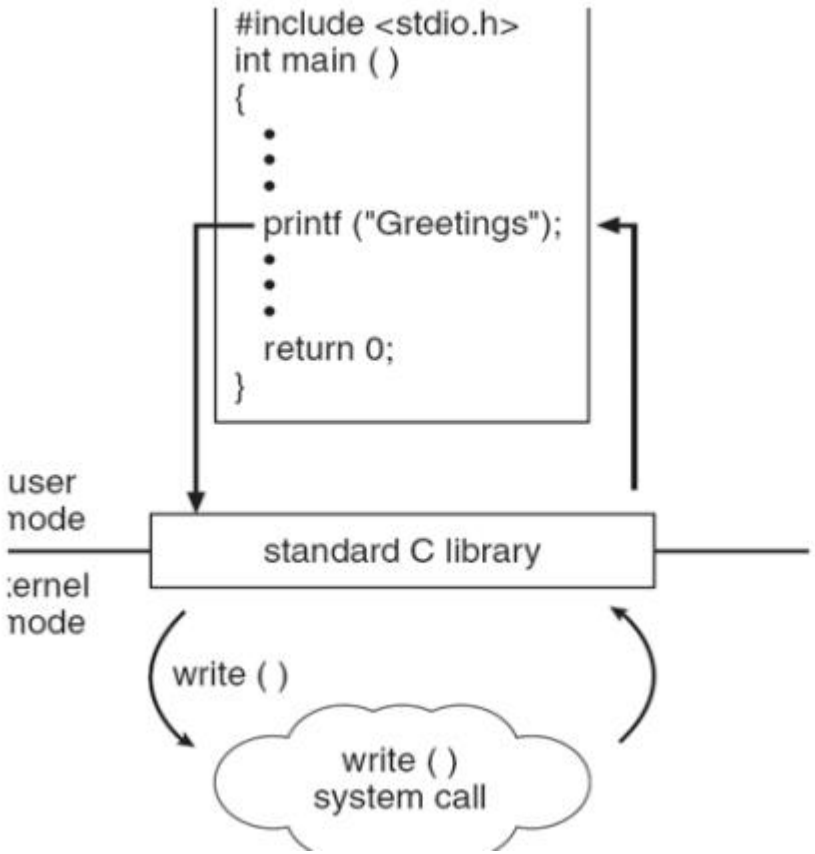
SYSTEM CALLS

- 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 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), and Java API for the Java virtual machine (JVM)

EXAMPLE OF HOW SYSTEM CALLS ARE USE



C Program Invoking Printf() Library Call, Which Calls Write() System Call



EXAMPLE OF STANDARD API

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>

ssize_t     read(int fd, void *buf, size_t count)
```

return value — function name — 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.

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()`

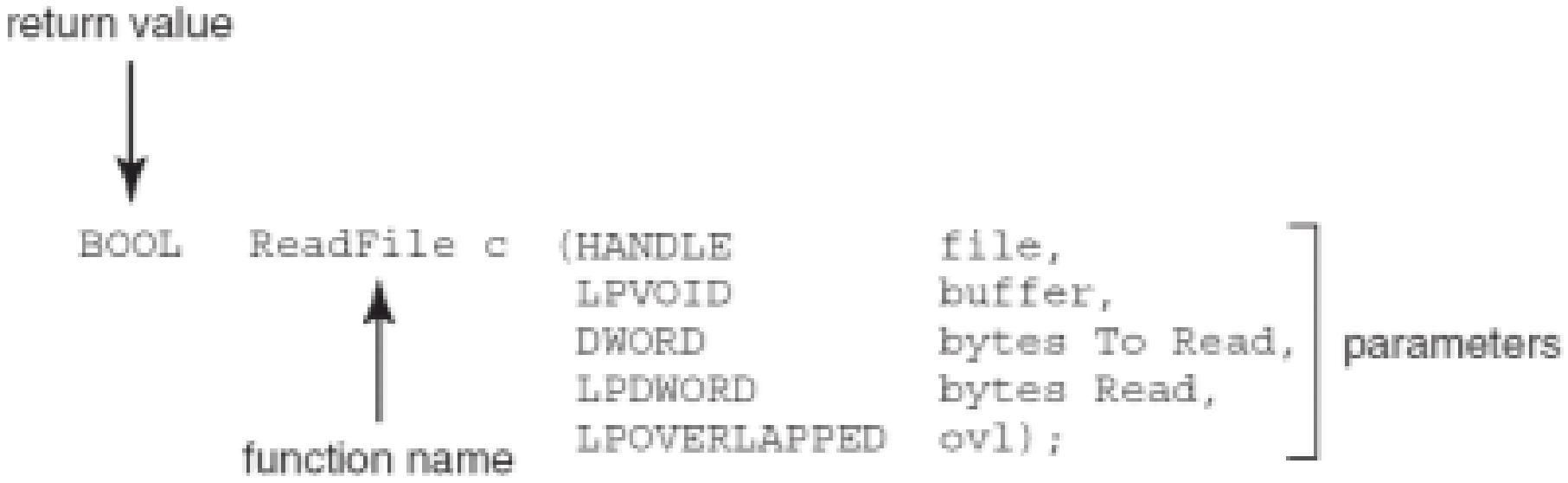
Device Manipulation	`SetConsoleMode()`
	`ReadConsole()`
	`WriteConsole()`
		`ioctl()`
		`read()`
		write()

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

Communication	`CreatePipe()`
	`CreateFileMapping()`
	`MapViewOfFile()`
		`pipe()`
		`shm_open()`
		`mmap()`

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

EXAMPLE OF STANDARD API



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

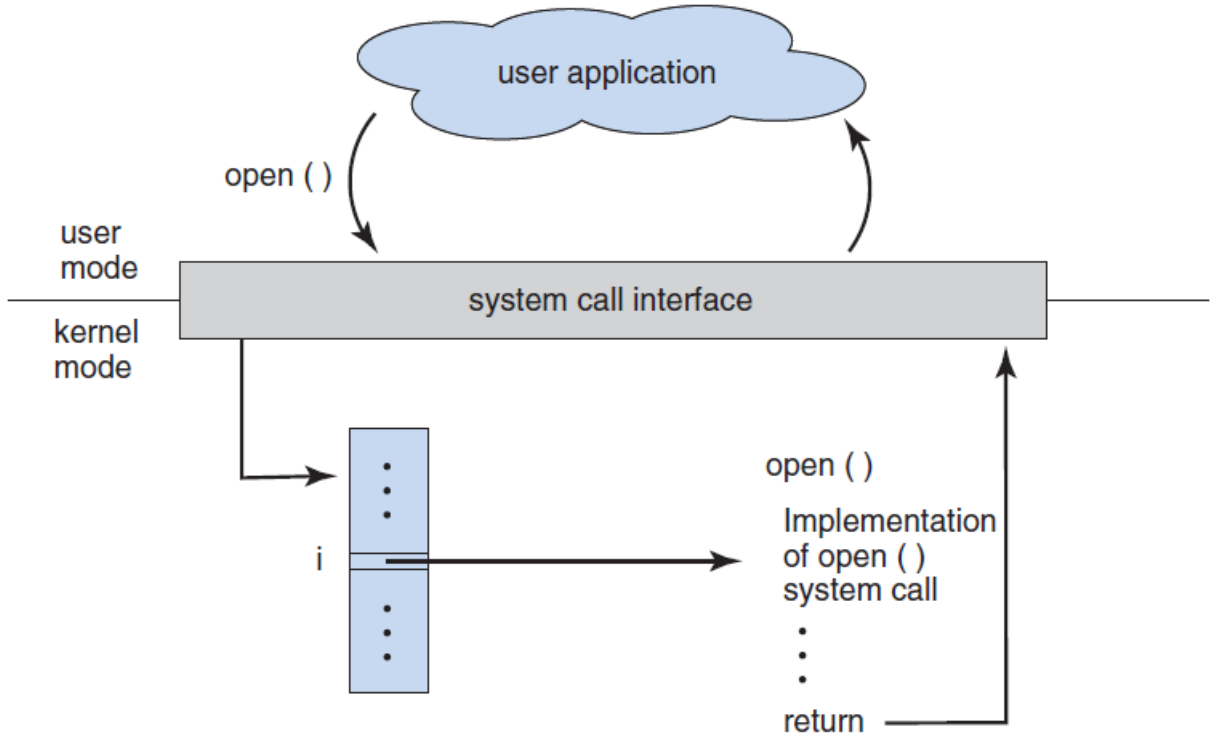
- The caller doesn't need to know about how the system call is implemented

  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)

How The Operating System Handles A User Application Invoking The open() System Call



System Call Parameter Passing

- Often, more information is required than simply identity of desired system call

  Exact type and amount of information vary according to OS and call

- Three general methods used to pass parameters to the OS

  1. Simplest: pass the parameters in *registers*
     - In some cases, may be more parameters than registers
  2. Parameters stored in a *block*, or table, in memory, and address of block passed as a parameter in a register
     - 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

- Block and stack methods do not limit the number or length of parameters being passed

THANK YOU