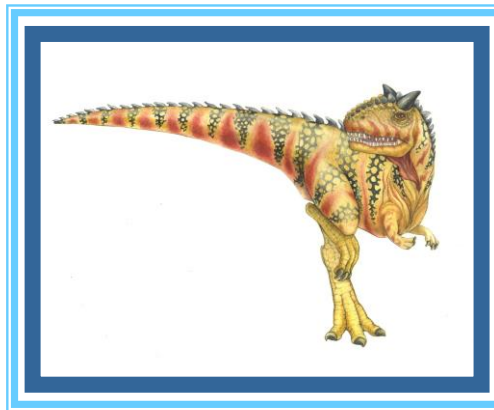
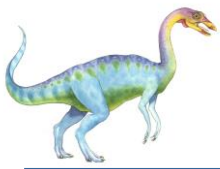
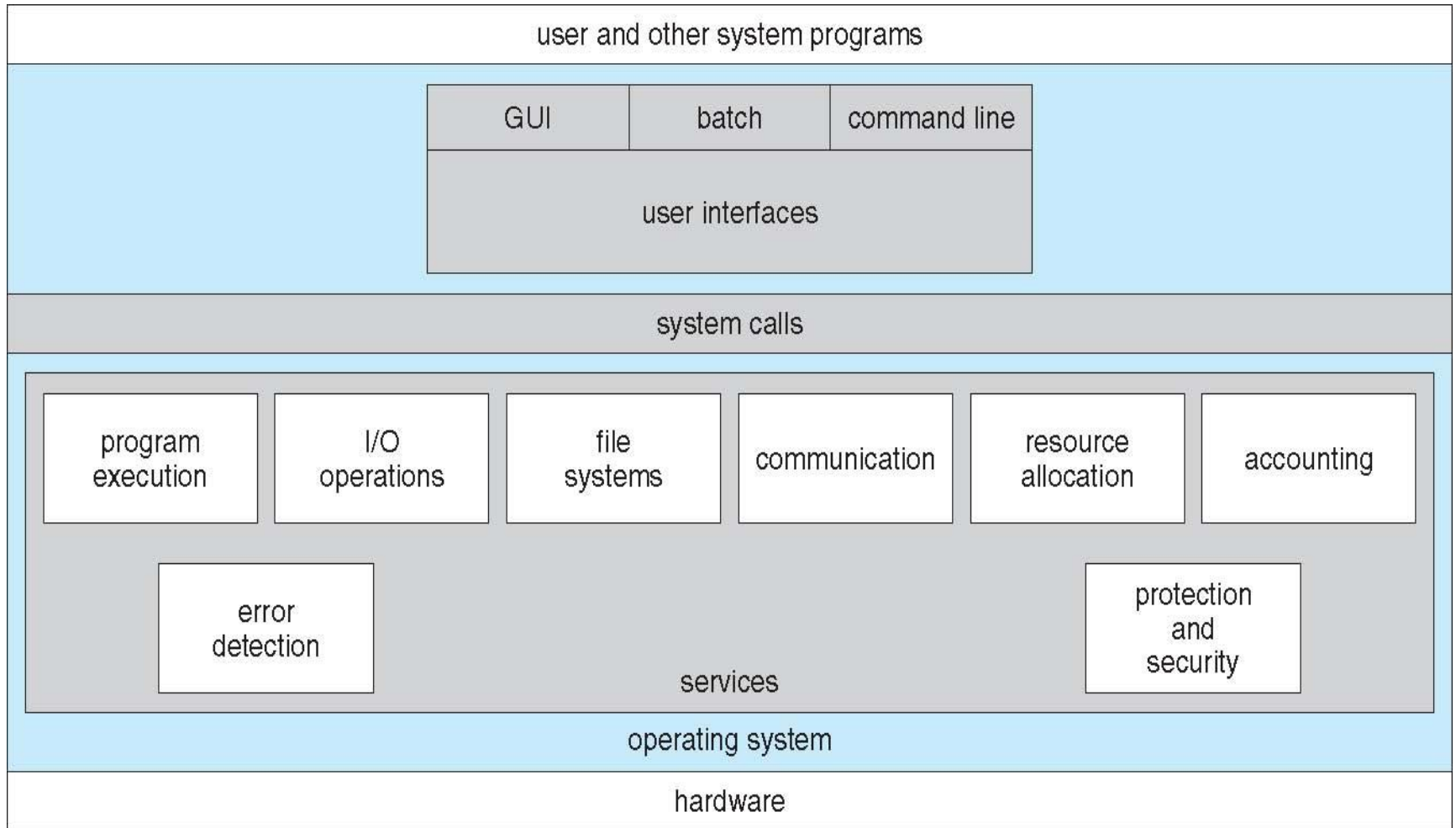


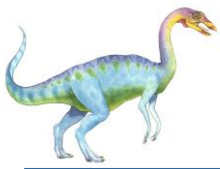
Chapter 2: System Structures





A View of Operating System Services





Operating System Services

- ❑ Operating systems provide an environment for execution of programs and services to programs and users
- ❑ One set of operating-system services provides functions that are helpful to the user:
 - ❑ **User interface** - Almost all operating systems 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 to 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

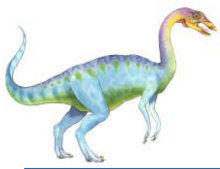




Operating System Services (Cont.)

- One set of operating-system services provides functions that are helpful to the user (Cont.):
 - **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)
 - **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

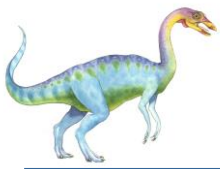




Operating System Services (Cont.)

- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing
 - **Resource 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
 - **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





User Operating System 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



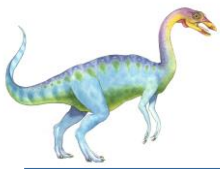


Bourne Shell Command Interpreter

```
Default
New Info Close Execute Bookmarks

PBG-Mac-Pro:~ pbg$ w
15:24 up 56 mins, 2 users, load averages: 1.51 1.53 1.65
USER      TTY      FROM          LOGIN@  IDLE WHAT
pbg       console -             14:34    50 -
pbg       s000    -             15:05    - w
PBG-Mac-Pro:~ pbg$ iostat 5
          disk0      disk1      disk10      cpu      load average
      KB/t tps MB/s  KB/t tps MB/s  KB/t tps MB/s  us sy id  1m  5m  15m
      33.75 343 11.30   64.31 14  0.88   39.67  0  0.02 11  5 84  1.51 1.53 1.65
      5.27 320  1.65    0.00  0  0.00    0.00  0  0.00  4  2 94  1.39 1.51 1.65
      4.28 329  1.37    0.00  0  0.00    0.00  0  0.00  5  3 92  1.44 1.51 1.65
^C
PBG-Mac-Pro:~ pbg$ ls
Applications          Music                  WebEx
Applications (Parallels)  Pando Packages       config.log
Desktop               Pictures              getsmartdata.txt
Documents             Public                imp
Downloads             Sites                 log
Dropbox               Thumbs.db             panda-dist
Library              Virtual Machines     prob.txt
Movies               Volumes               scripts
PBG-Mac-Pro:~ pbg$ pwd
/Users/pbg
PBG-Mac-Pro:~ pbg$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: icmp_seq=0 ttl=64 time=2.257 ms
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=1.262 ms
^C
--- 192.168.1.1 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 1.262/1.760/2.257/0.498 ms
PBG-Mac-Pro:~ pbg$
```

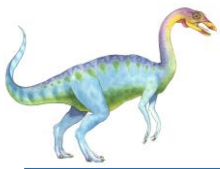




User Operating System Interface - GUI

- ❑ 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 directory (known as a **folder**))
 - ❑ Invented at Xerox PARC
- ❑ 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)





Touchscreen Interfaces

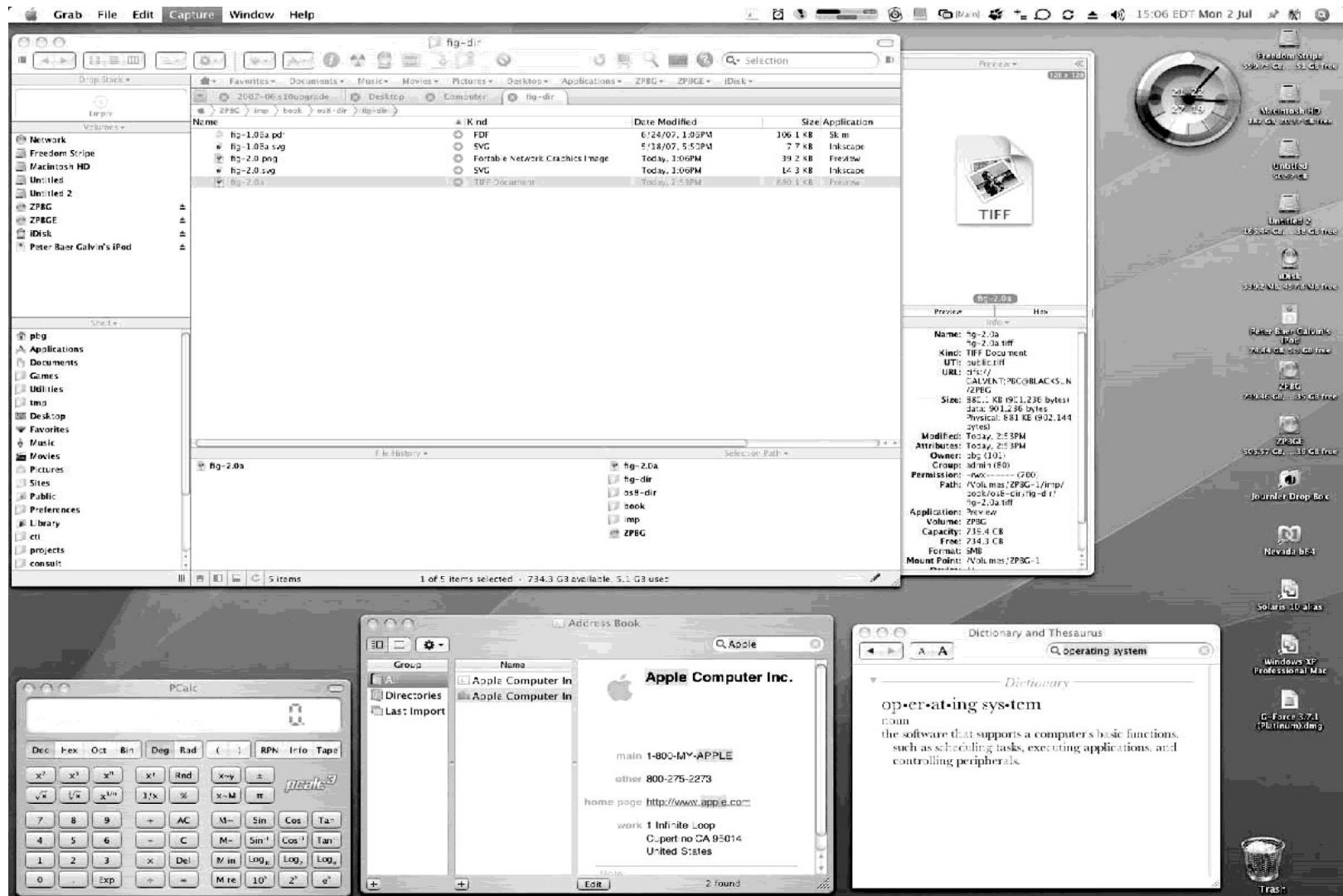
- n Touchscreen devices require new interfaces
 - | Mouse not possible or not desired
 - | Actions and selection based on gestures
 - | Virtual keyboard for text entry
 - | Voice commands.

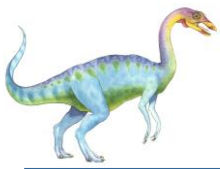
Haptic





The Mac OS X GUI





System Calls

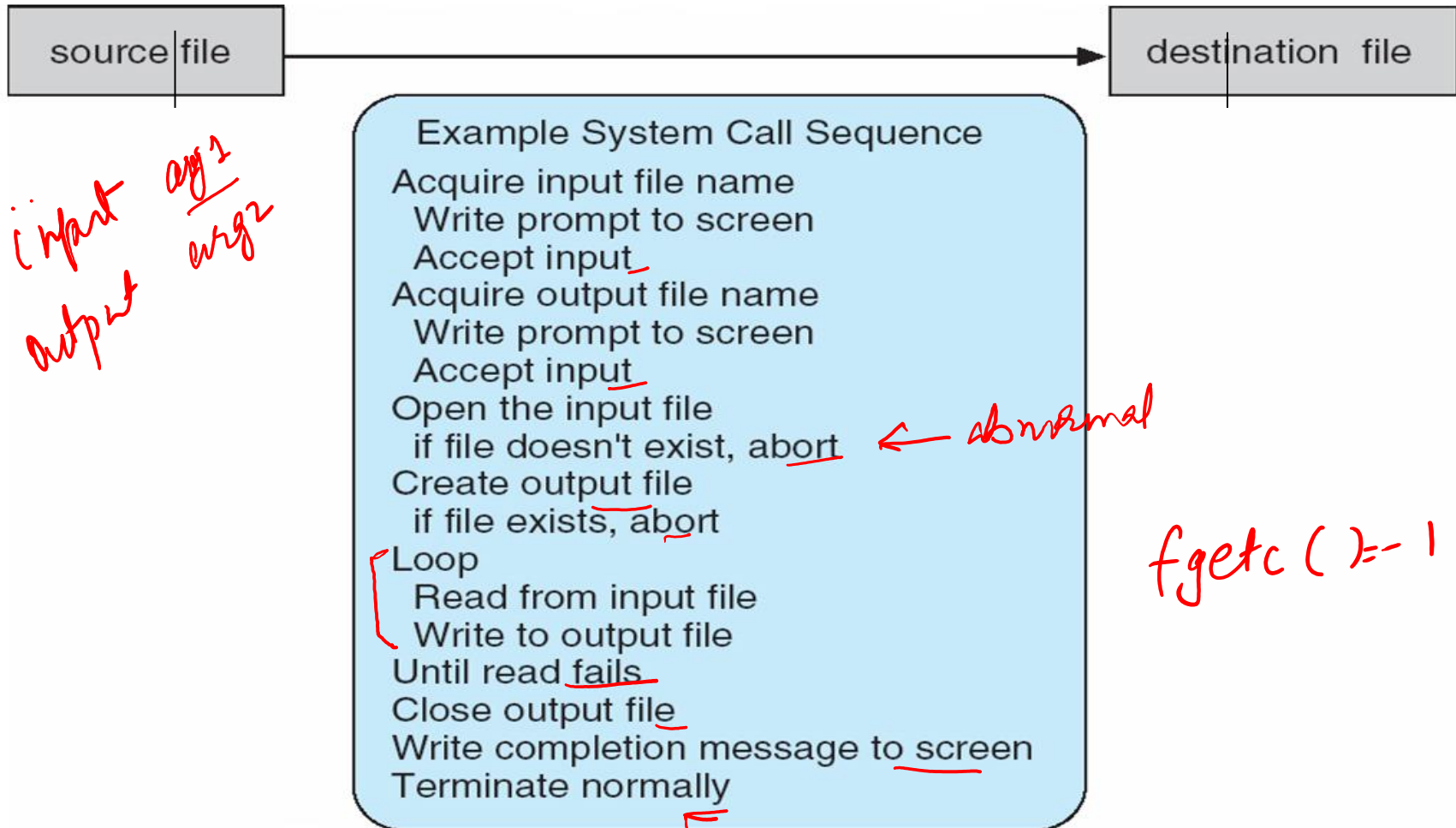
- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Programming Interface (API)** rather than direct system call use
- Three most common APIs are Win32 API for Windows, POSIX API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

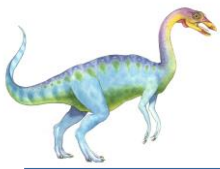




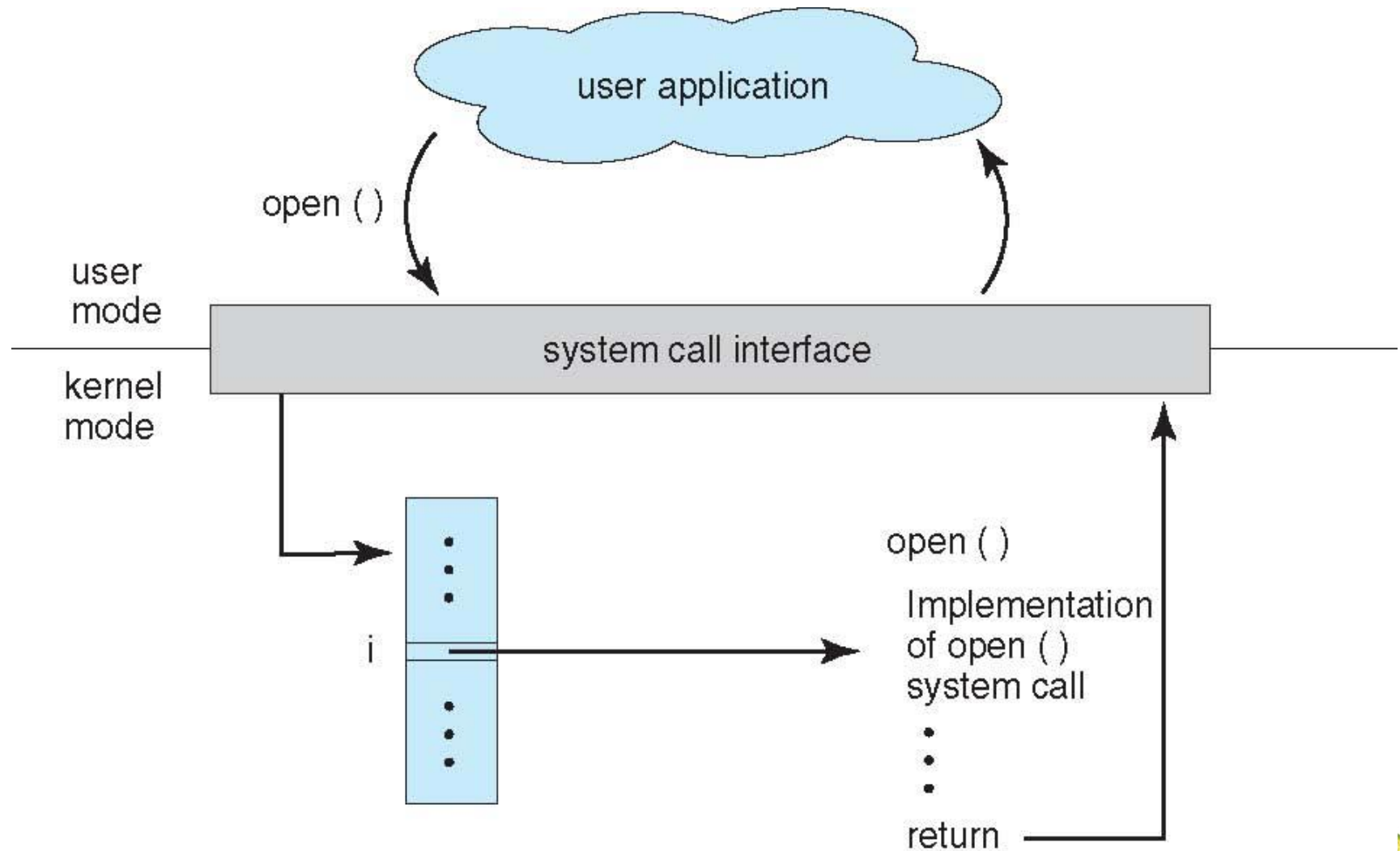
System Call Implementation

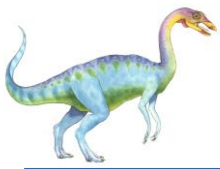
- Typically, a number associated with each system call
 - **System-call interface** maintains a table indexed according to these numbers
- The system call interface invokes the intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing 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





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
 - Simplest: pass the parameters in registers
 - ▶ In some cases, 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 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





Parameter Passing via Table

