

UNIT I

INTRODUCTION



Operating Systems



Introduction

- **Introduction**

- What Operating Systems Do
- Computer-System Architecture
- Operating-System Structure
- Operating-System Operations
- Operating-System Services
- User Operating System Interface
- **System Calls**
- **Types of System Calls**
- System Programs
- System Boot

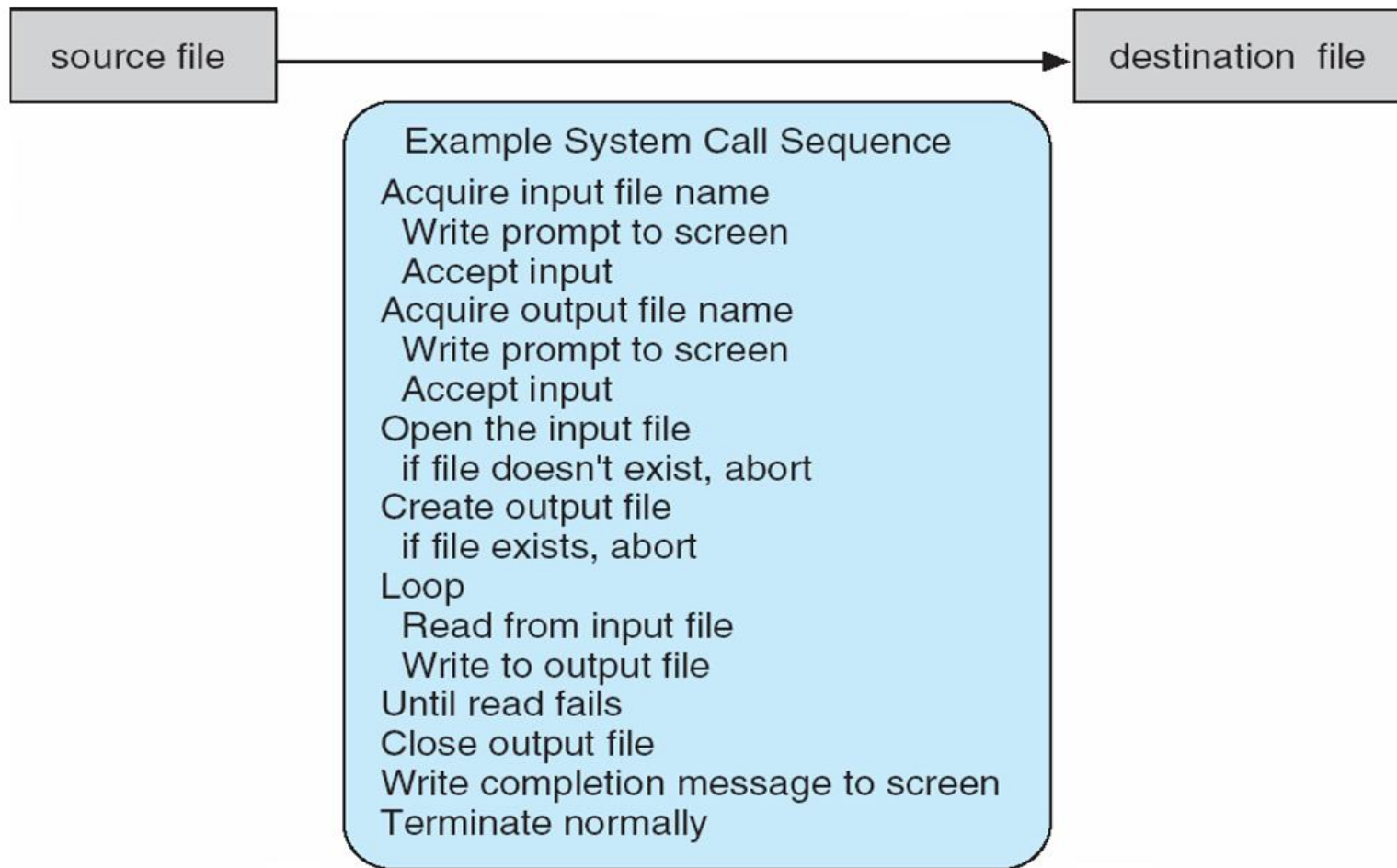
- **Process Concept**

- Process Scheduling
- Operations on Processes
- Interprocess Communication

- **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

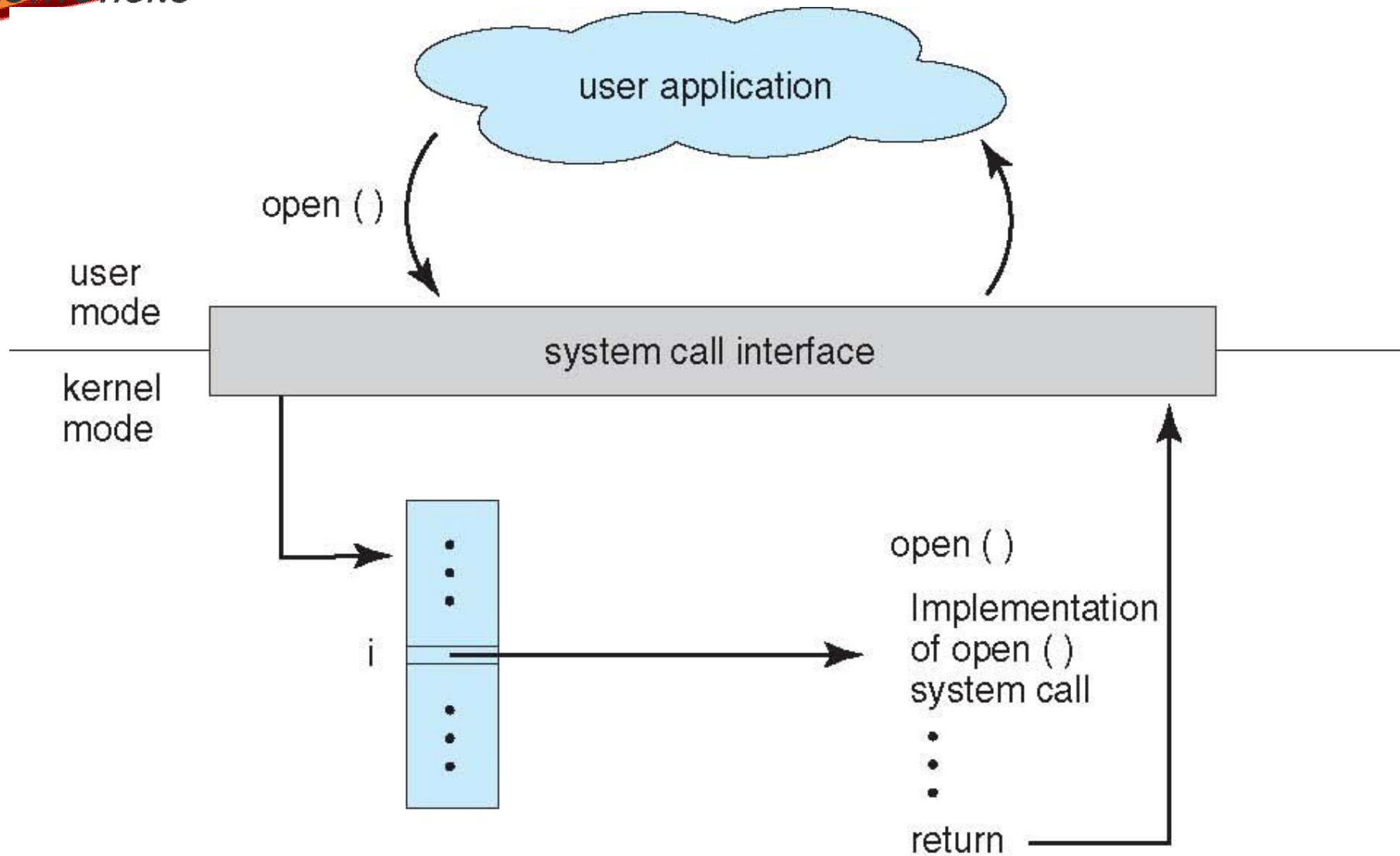
On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns `-1`.



- 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



API - System Call - OS Relationship



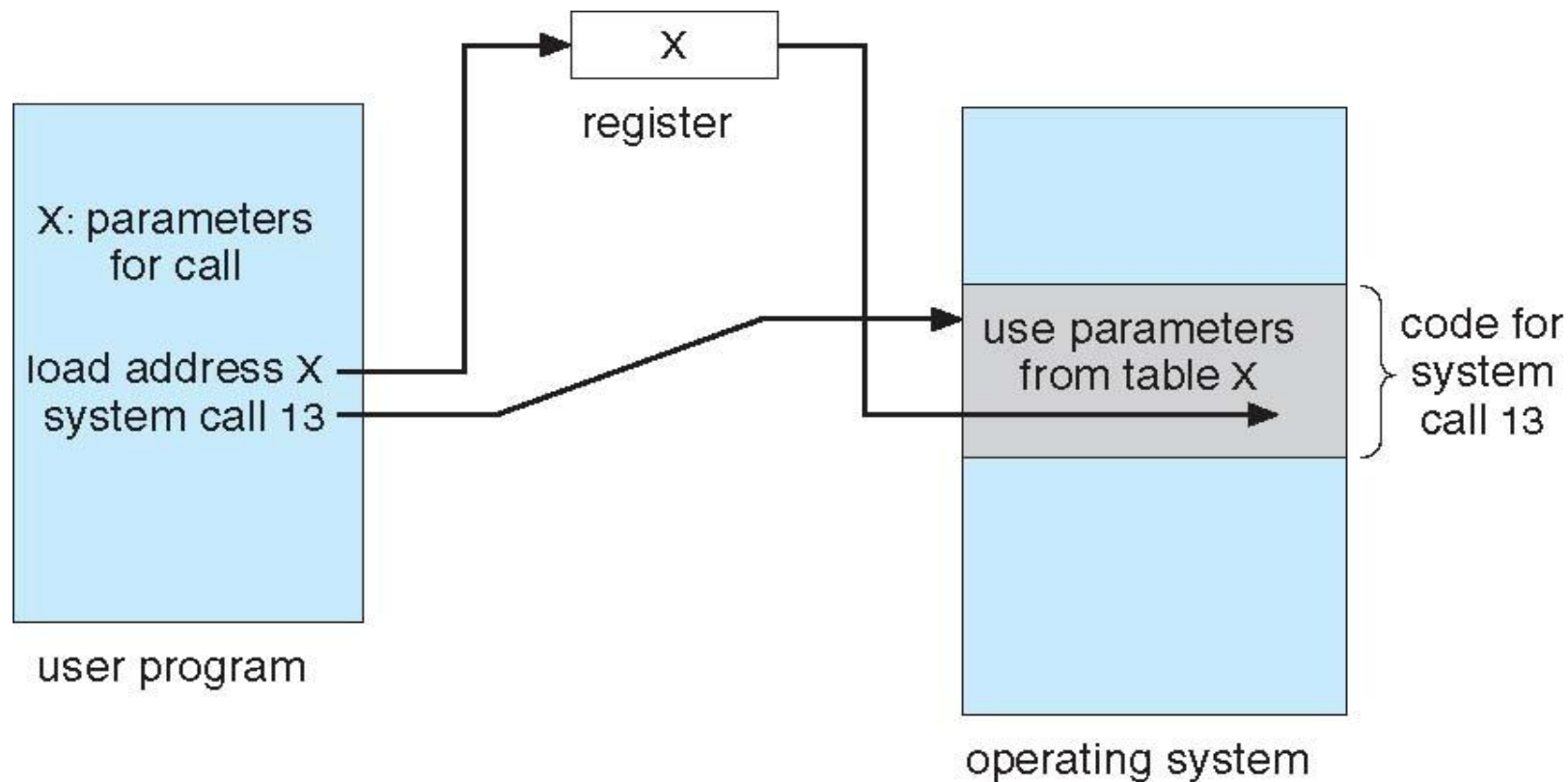


System Call Parameter Passing

- **Three general methods** used to pass parameters to the OS
 - **Simplest:** pass the parameters in 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



Parameter Passing via Table





Types of System Calls

- **Process control**

- create process, terminate process
- end, abort , load, execute
- get process attributes, set process attributes
- wait for time , wait event, signal event
- allocate and free memory
- Dump memory if error
- **Debugger** for determining **bugs, single step** execution
- **Locks** for managing access to shared data between processes

- **File management**

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

- **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 if **message passing model** to **host name** or **process name**
- **Shared-memory model** create and gain access to memory regions
- transfer status information
- attach and detach remote devices

Protection

Control access to resources
Get and set permissions
Allow and deny user access



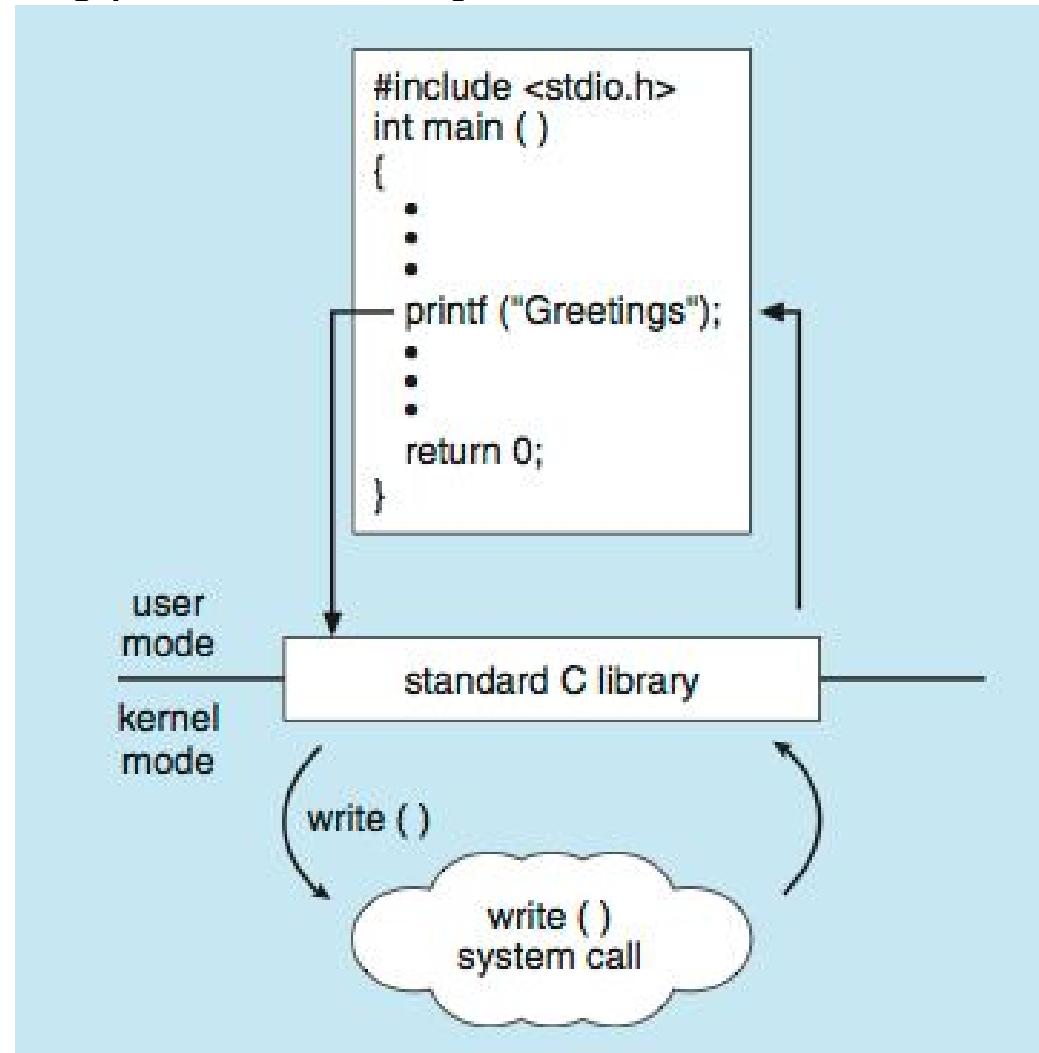
Examples of Windows and Unix System Calls

	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() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()



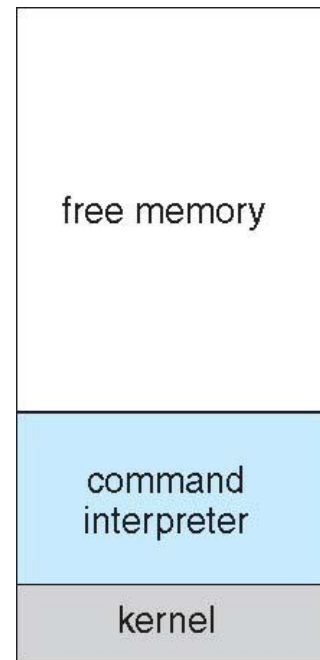
Standard C Library Example

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



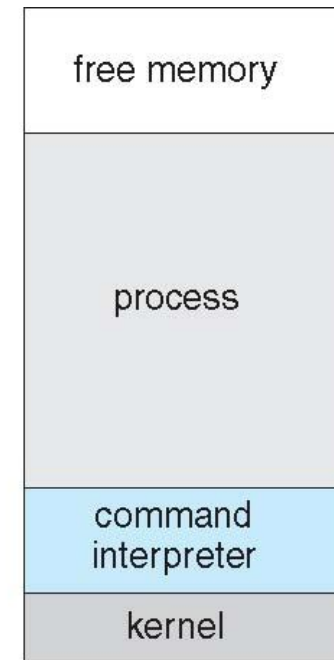
Example: MS-DOS

- Single-tasking
- Shell invoked when system booted
- Simple method to run program
 - No process created
- Single memory space
- Loads program into memory, overwriting all but the kernel
- Program exit -> shell reloaded



(a)

At system startup

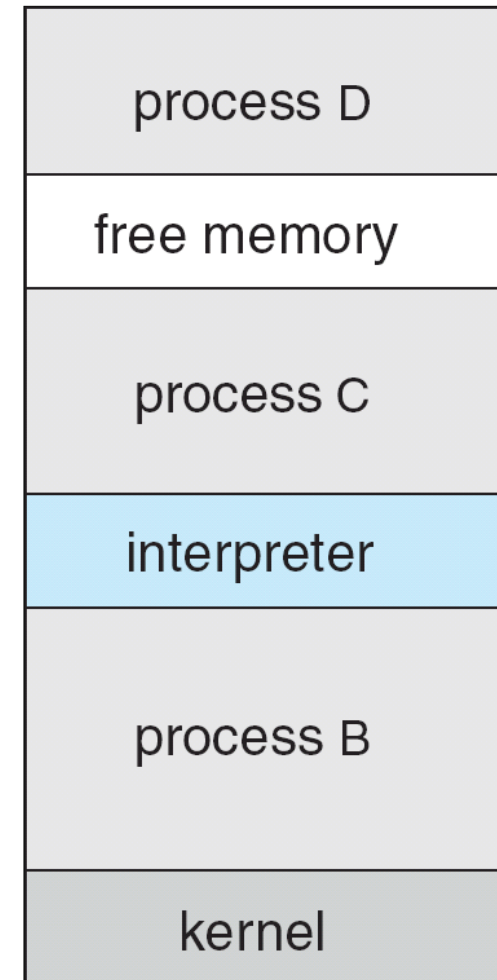


(b)

running a program

Example: FreeBSD

- Unix variant , Multitasking
- User login -> invoke user's choice of shell
- Shell executes **fork()** system call to create process
 - Executes **exec()** to load program into process
 - Shell waits for process to terminate or continues with user commands
- Process exits with:
 - code = 0 – no error
 - code > 0 – error code





TEXT BOOK

1. Abraham Silberschatz, Peter B. Galvin, "Operating System Concepts", 10th Edition, John Wiley & Sons, Inc., 2018.
2. Jane W. and S. Liu. "Real-Time Systems". Prentice Hall of India 2018.
3. Andrew S Tanenbaum, Herbert Bos, Modern Operating Pearson , 2015.

REFERENCES

1. William Stallings, "Operating Systems: Internals and Design Principles", 9th Edition, Prentice Hall of India., 2018.
2. D.M.Dhamdhere, "Operating Systems: A Concept based Approach", 3rd Edition, Tata McGraw hill 2016.
3. P.C.Bhatt, "An Introduction to Operating Systems–Concepts and Practice", 4th Edition, Prentice Hall of India., 2013.

THANK YOU