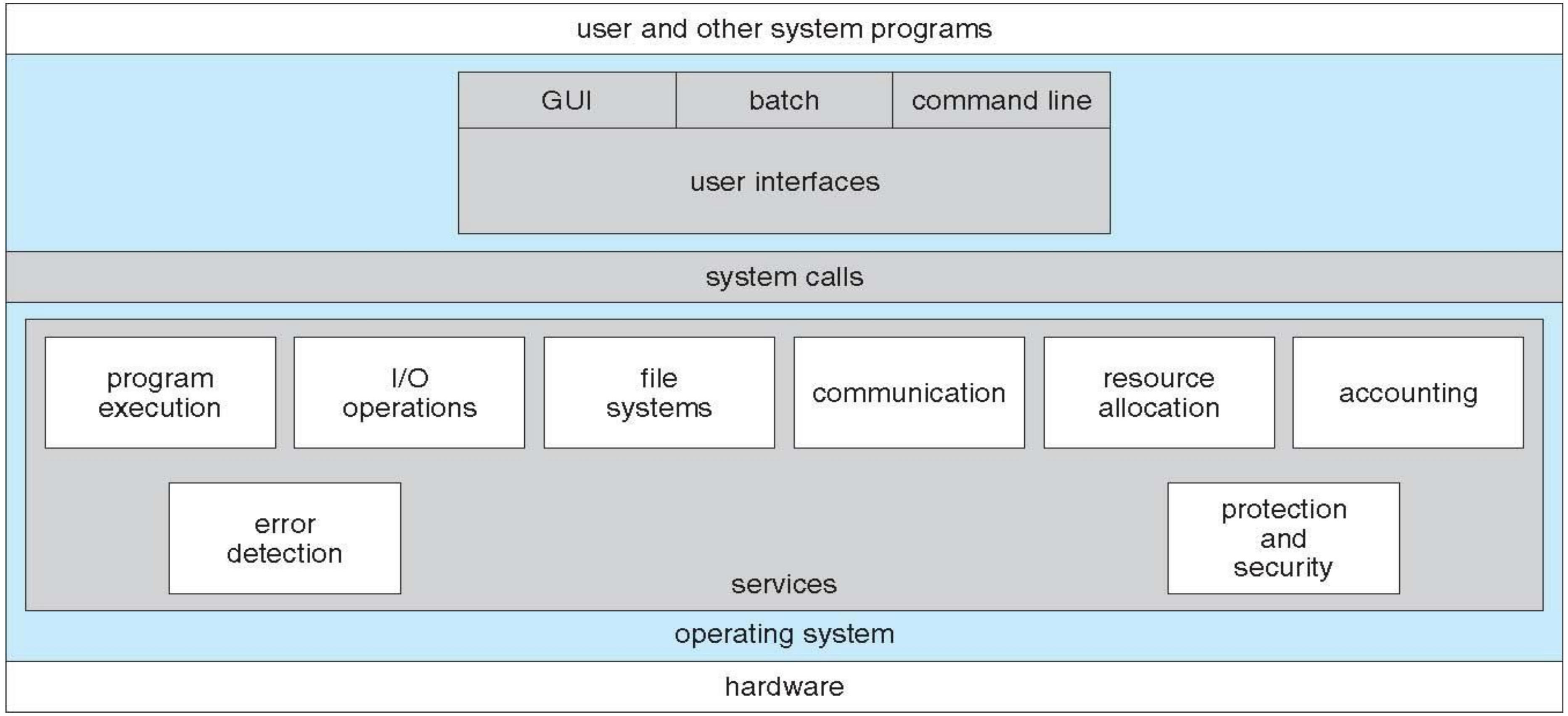
System structure

2.1 Operating system services

- An operating system provides an environment for the execution of programs. It provides certain services to programs and to the users of those programs. The specific services provided, of course, differ from one operating system to another, but we can identify common classes. Following figure shows one view of the various operating-system services and how they interrelate.



Continued….

- User interface. Almost all operating systems have a user interface. This interface can take several forms. One is a DTrace command line interface(CLI) which uses text commands and a method for entering them(say, a program to allow entering and editing of commands). Another is a batch interface in which commands and directives to control those commands are entered into files, and those files are executed. Most commonly, a graphical user interface is used.
- Program execution. The system must be able to load a program into memory and to run that program. The program must be able to end its execution, either normally or abnormally (indicating error).
- I/O operations. A running program may require I/0, which may involve a file or an I/0 device. For specific devices, special functions may be desired (such as recording to a CD or DVD drive or blanking a display screen). For efficiency and protection, users usually cannot control I/0 devices directly. Therefore, the operating system must provide a means to do I/0.
- File-system manipulation. The file system is of particular interest. Obviously, programs need to read and write files and directories. They also need to create and delete them by name, search for a given file, and list file information. Finally, some programs include permissions management to allow or deny access to files or directories based on file ownership.

Continued….

- Communications. There are many circumstances in which one process needs to exchange information with another process. Such communication may occur between processes that are executing on the same computer or between processes that are executing on different computer systems tied together by a computer network. **Communications may be implemented via *shared memory* or through *message passing.***
- Error detection. The operating system needs to be constantly aware of possible errors. Errors may occur in the CPU and memory hardware (such as a memory error or a power failure), in I/0 devices (such as a parity error on tape, a connection failure on a network, or lack of paper in the printer), and in the user program (such as an arithmetic overflow, an attempt to access an illegal memory location). For each type of error, the operating system should take the appropriate action to ensure correct and consistent computing.

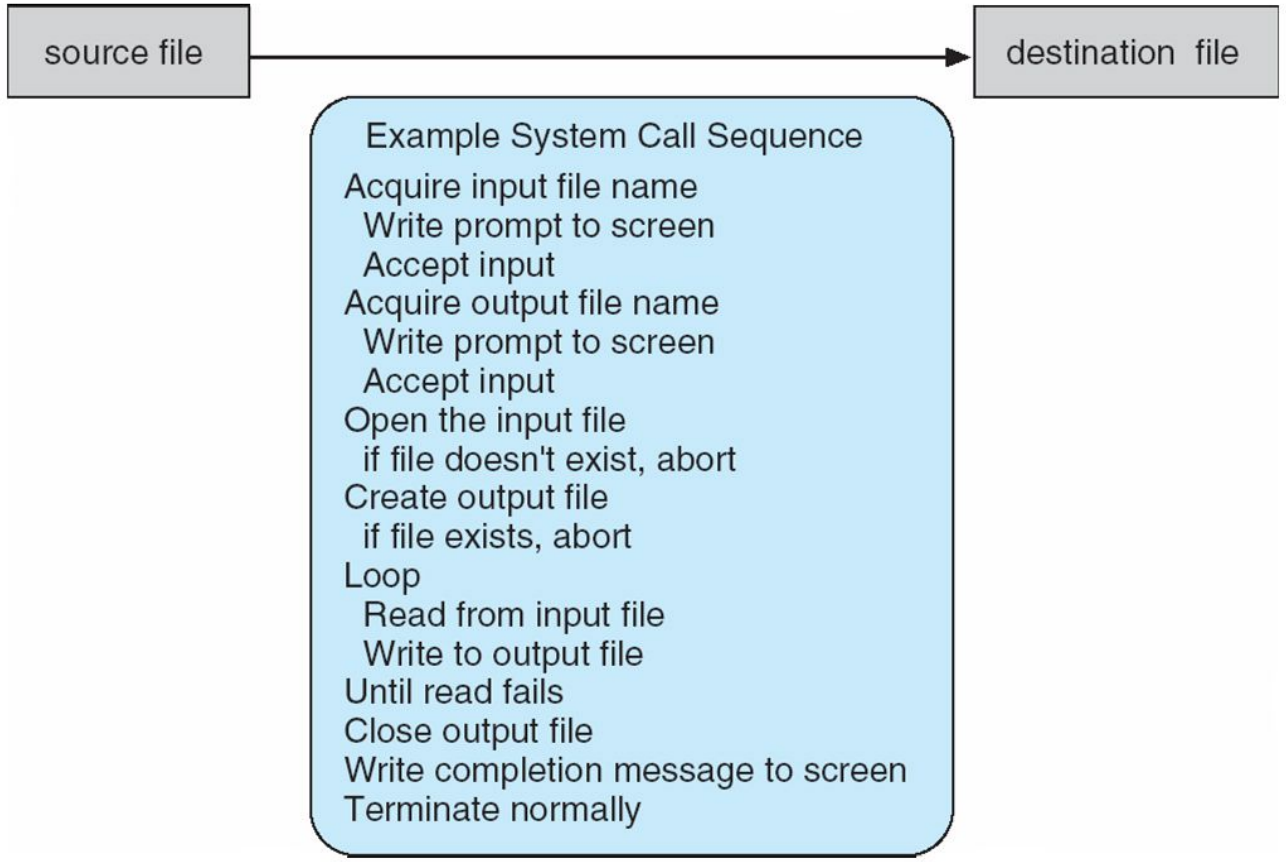
Continued…

Another set of operating-system functions exists not for helping the user but rather for ensuring the efficient operation of the system itself.

- Resource allocation: When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them. Many different -types of resources are managed by the operating system such as CPU cycles, main memory, and file storage.
- Accounting: we want to keep track of which users use how much and what kinds of computer resources. This record keeping may be used for accounting or simply for accumulating usage statistics.
- Security of the system from outsiders is also important. Such security starts with requiring each user to authenticate himself or herself to the system, usually by means of a password, to gain access to system resources.

2.3 system call

- System calls provide an interface to the services made available by an operating system. These calls are generally available as routines written in C and C++.
- System call sequence to copy the contents of one file to another file.



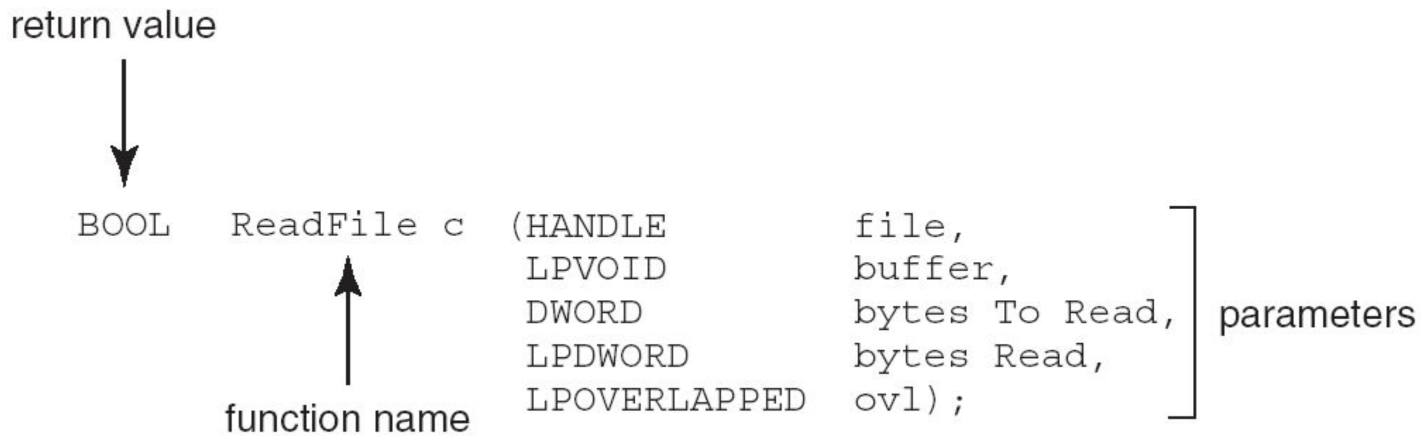
```
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```

Continued….

- **Application programming interface(API) :** specifies a set of functions that are available to application programmer, including the parameters that are passed to each function and the return values the programmer can expect.

Example of Standard API

- Consider the ReadFile() function in the
- Win32 API—a function for reading from a file

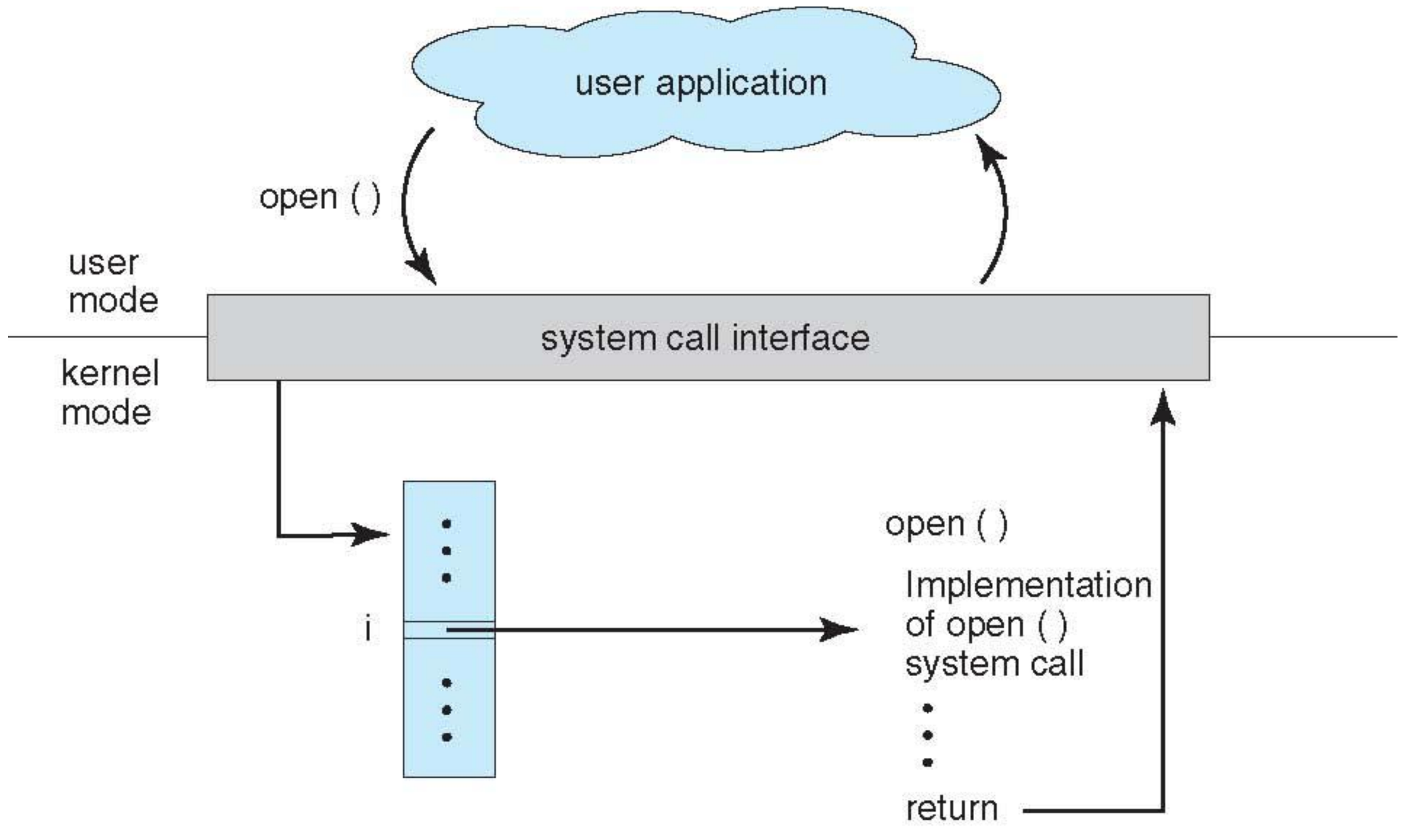


```
BOOL   ReadFile c  (HANDLE          file,
                    LPVOID          buffer,
                    DWORD           bytes To Read,
                    LPDWORD         bytes Read,
                    LPOVERLAPPED    ovl);
```

- A description of the parameters passed to ReadFile()
  - HANDLE file—the file to be read
  - LPVOID buffer—a buffer where the data will be read into and written from
  - DWORD bytesToRead—the number of bytes to be read into the buffer
  - LPDWORD bytesRead—the number of bytes read during the last read
  - LPOVERLAPPED ovl—indicates if overlapped I/O is being used

Continued…

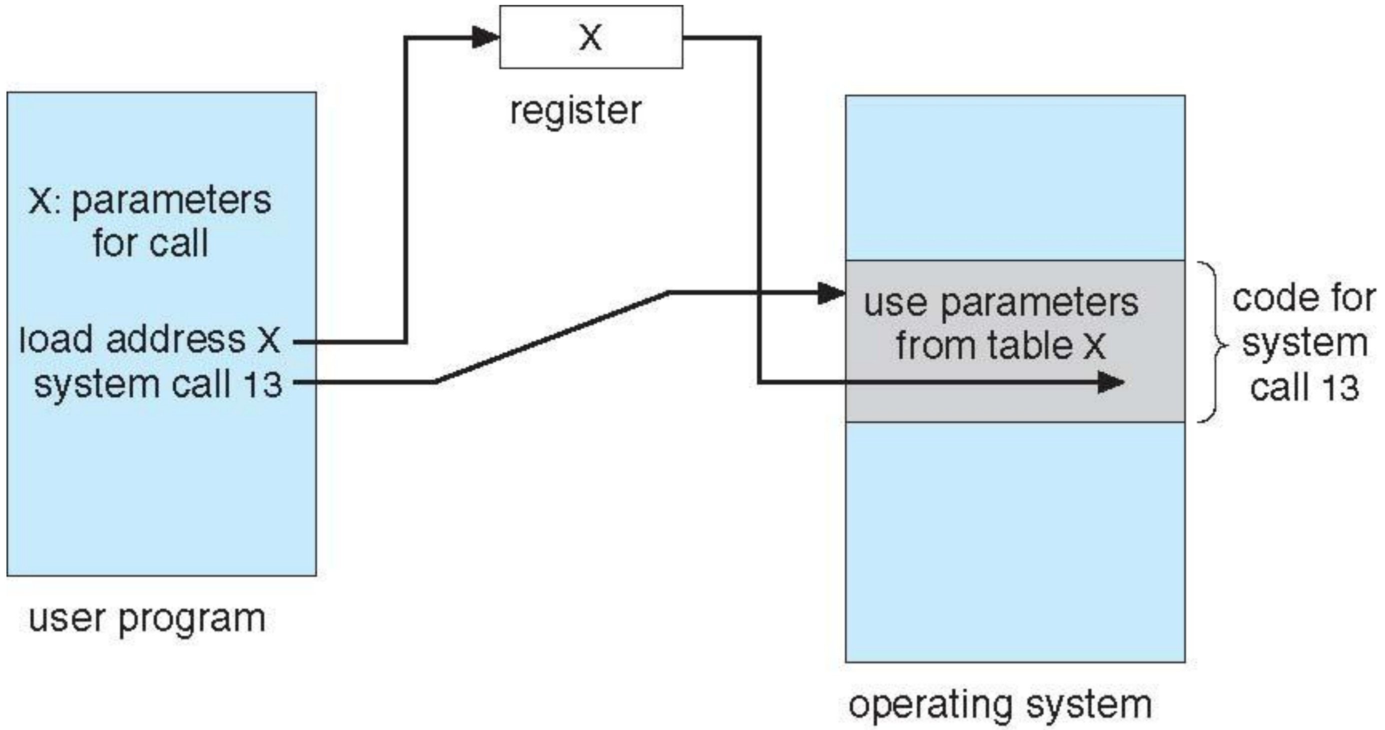
- **The relationship between an API, the system-call interface, and the operating system is shown in** Figure 2.6, which illustrates how the operating system handles a user application invoking the open() system call.



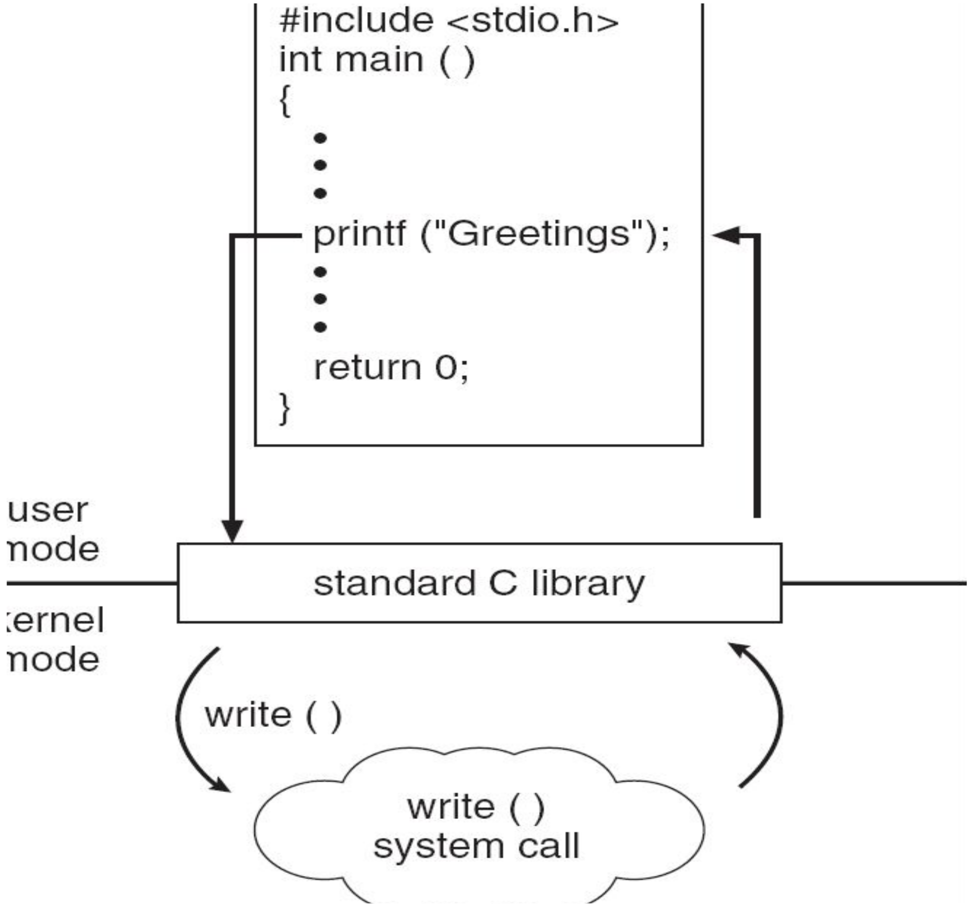
System Call Parameter Passing

- 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



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



```
#include <stdio.h>
int main ( )
{
   •
   •
   •
   printf ("Greetings");
   •
   •
   •
   return 0;
}
```

2.4 Types of system calls

System calls can be grouped roughly six major categories:

- Process control
- file manipulation
- Device manipulation
- information maintenance
- Communication
- Protection

Continued…

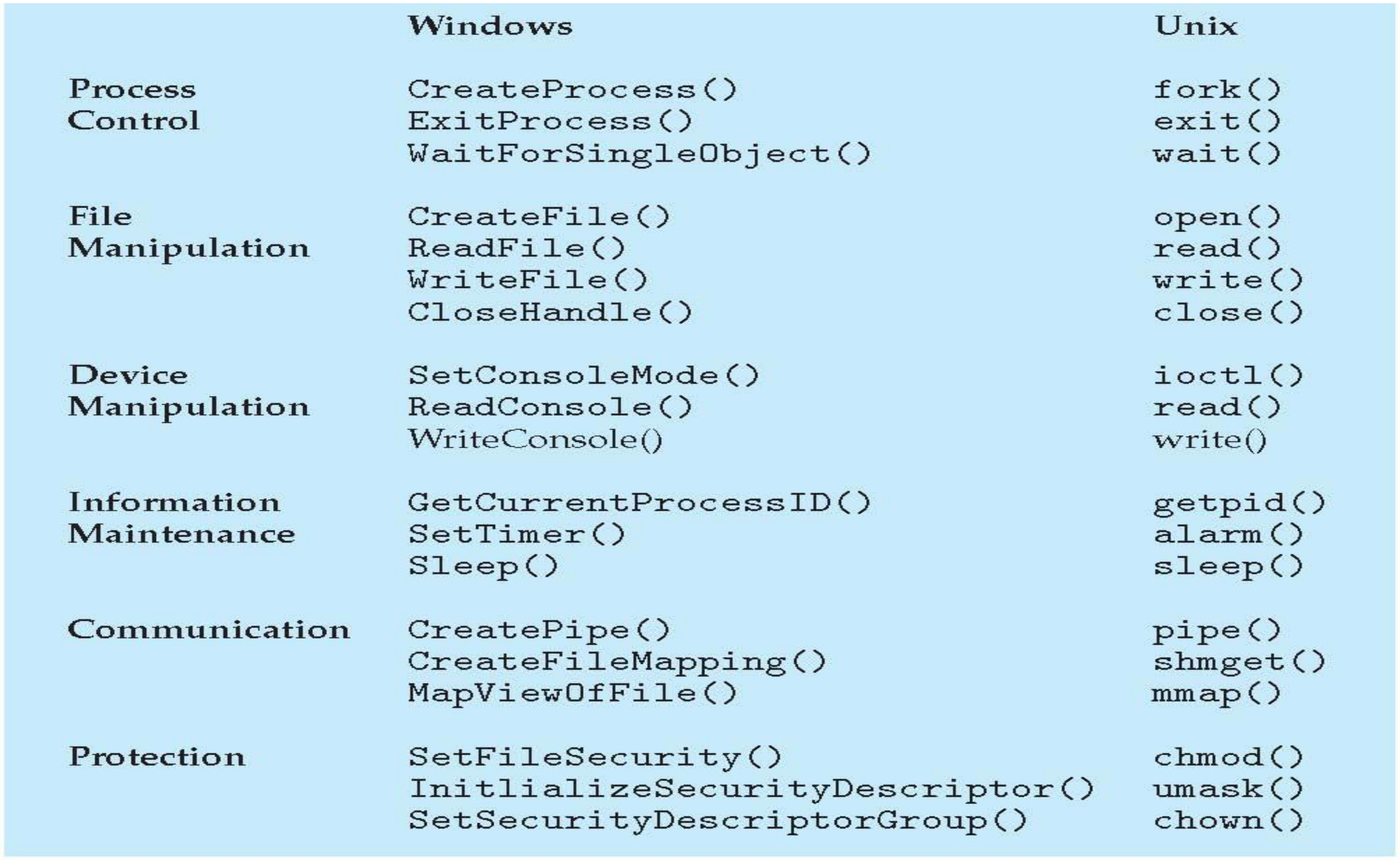
- Process control
  - End process, abort process
  - Load process, execute process
  - create process, terminate process
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
- File management
  - create file, delete file
  - open, close file
  - read, write, reposition
  - get and set file attributes

Continued…

- 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

Continued….

Examples of Windows and Unix System Calls

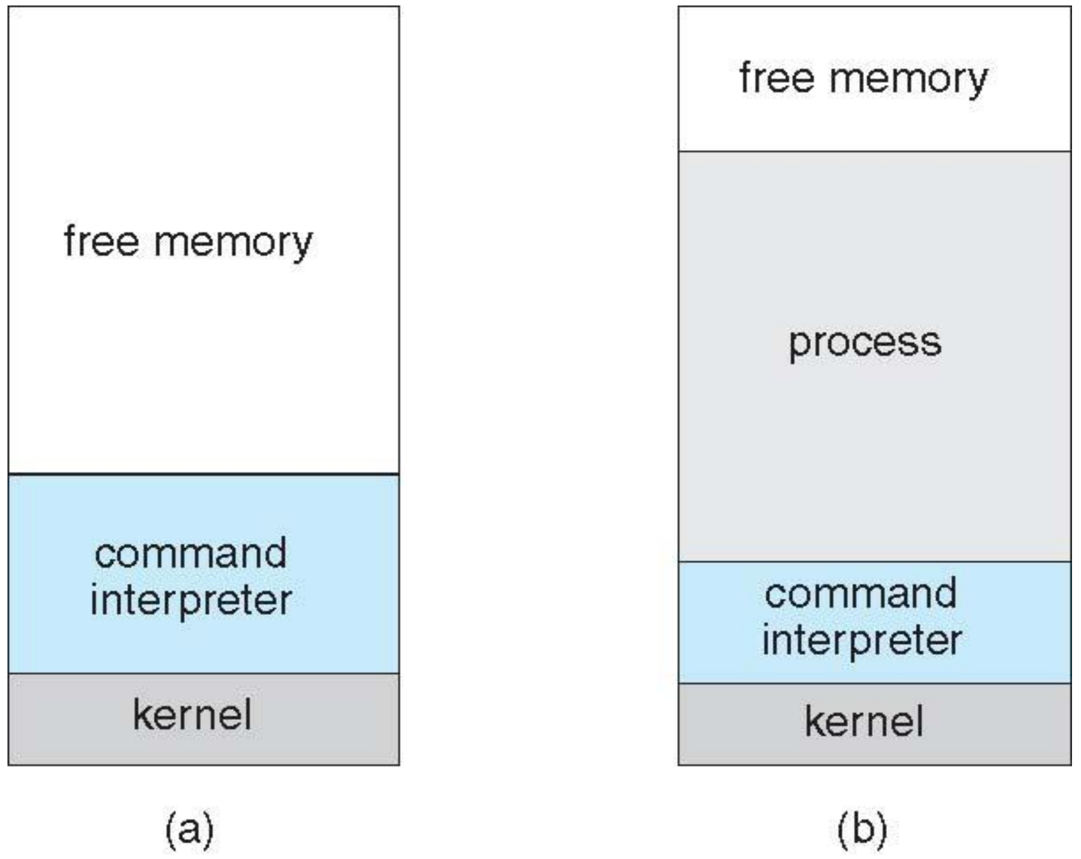


	Windows	Unix
Process Control	`CreateProcess()`	`fork()`
	`ExitProcess()`	`exit()`
	`WaitForSingleObject()`	`wait()`
File Manipulation	`CreateFile()`	`open()`
	`ReadFile()`	`read()`
	`WriteFile()`	`write()`
	`CloseHandle()`	`close()`
Device Manipulation	`SetConsoleMode()`	`ioctl()`
	`ReadConsole()`	`read()`
	`WriteConsole()`	`write()`
Information Maintenance	`GetCurrentProcessID()`	`getpid()`
	`SetTimer()`	`alarm()`
	`Sleep()`	`sleep()`
Communication	`CreatePipe()`	`pipe()`
	`CreateFileMapping()`	`shmget()`
	`MapViewOfFile()`	`mmap()`
Protection	`SetFileSecurity()`	`chmod()`
	`InitlializeSecurityDescriptor()`	`umask()`
	`SetSecurityDescriptorGroup()`	`chown()`

Continued….

- Example: MS-DOS
- Single tasking operating system.

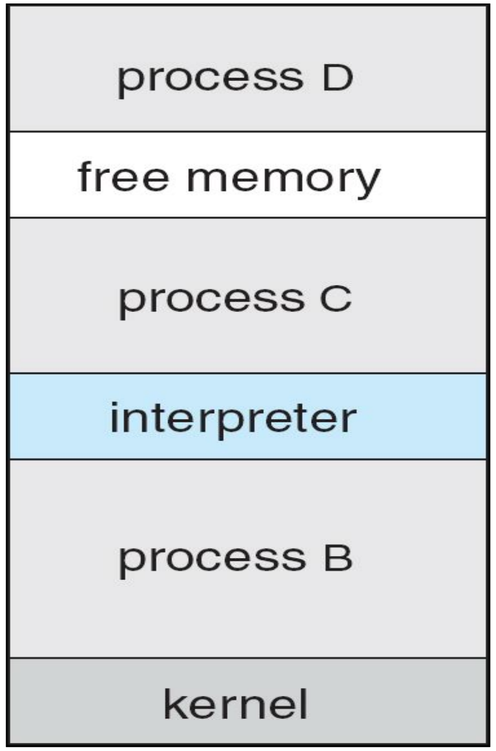
MS-DOS execution (a)at system startup (b)running a program



Continued…..

- **FreeBSD**
- Unix variant
- Multitasking
- User login -> invoke user's choice of shell
- To start new process Shell executes fork() system call to create process and exec() system call to load process.

FreeBSD Running Multiple Programs



2.7 Operating system structure

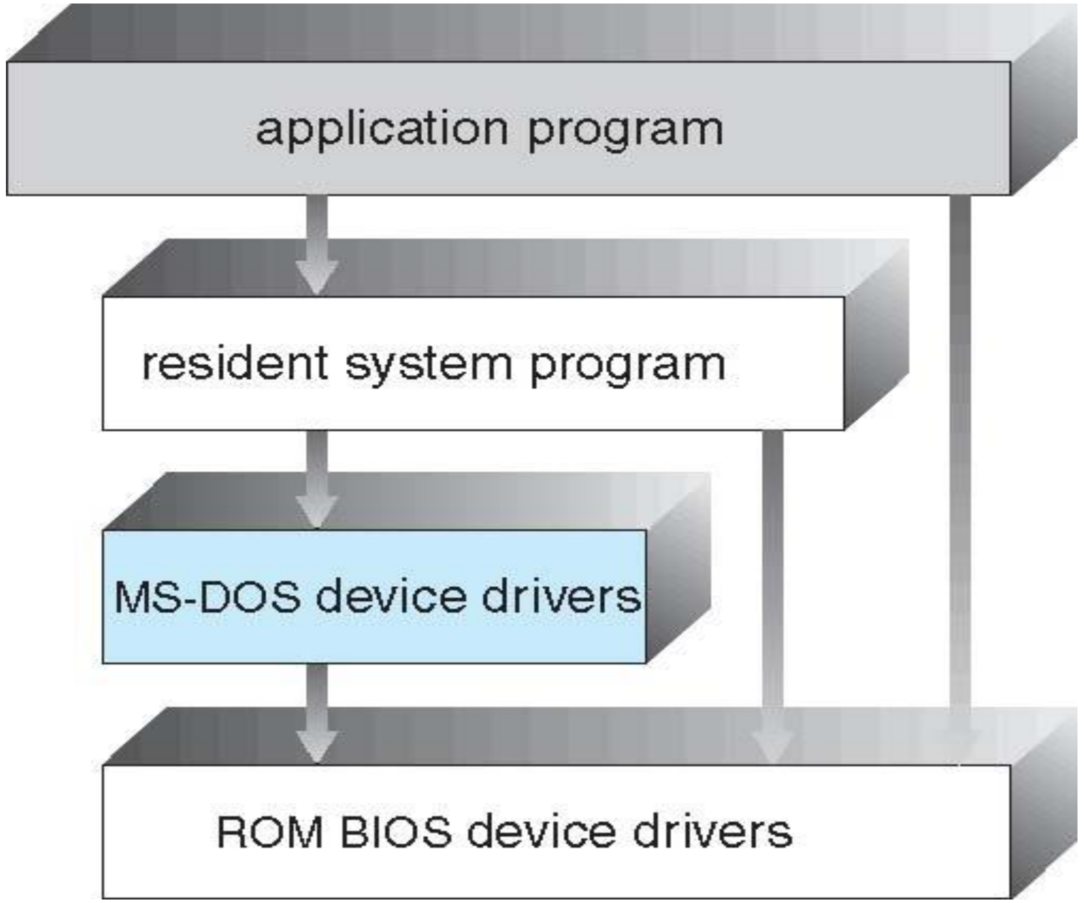
- A system as large and complex as a modern operating system must be engineered carefully if it is to function properly and be modified easily. A common approach is to partition the task into small components rather than have one monolithic system.

1. Simple structure 2. layered 3. microkernel 4. modular system

2.7.1 simple structure

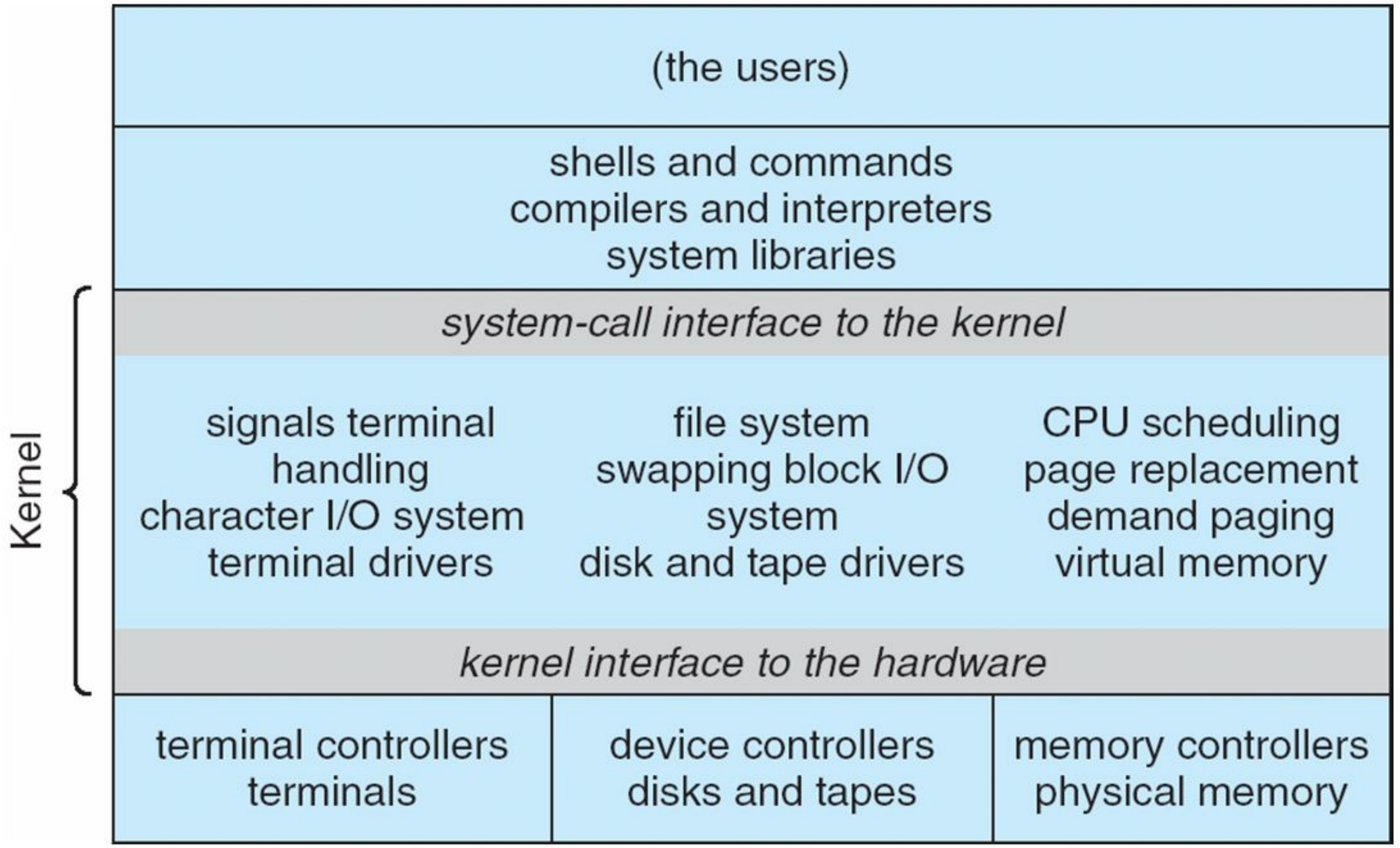
- Many commercial operating systems do not have well-defined structures. Frequently, such systems started as small, simple, and limited systems and then grew beyond their original scope.
- MS-DOS is an example of such a system.
- It was written to provide the most functionality in the least space, so it was not divided into modules carefully. For example application programs are able to access I/O routines to write directly to display devices and disk drives but entire system crashes if program fails. Following figure shows its structure.

Continued…


application program
resident system program
MS-DOS device drivers
ROM BIOS device drivers

Continued….

- Another example of limited structuring is the original UNIX operating system. Like MS-DOS, UNIX initially was limited by hardware functionality.
- It consists of two separate parts the kernel and system programs.
- **Figure shows Traditional UNIX System Structure.**



Continued….

Advantage of simple structure

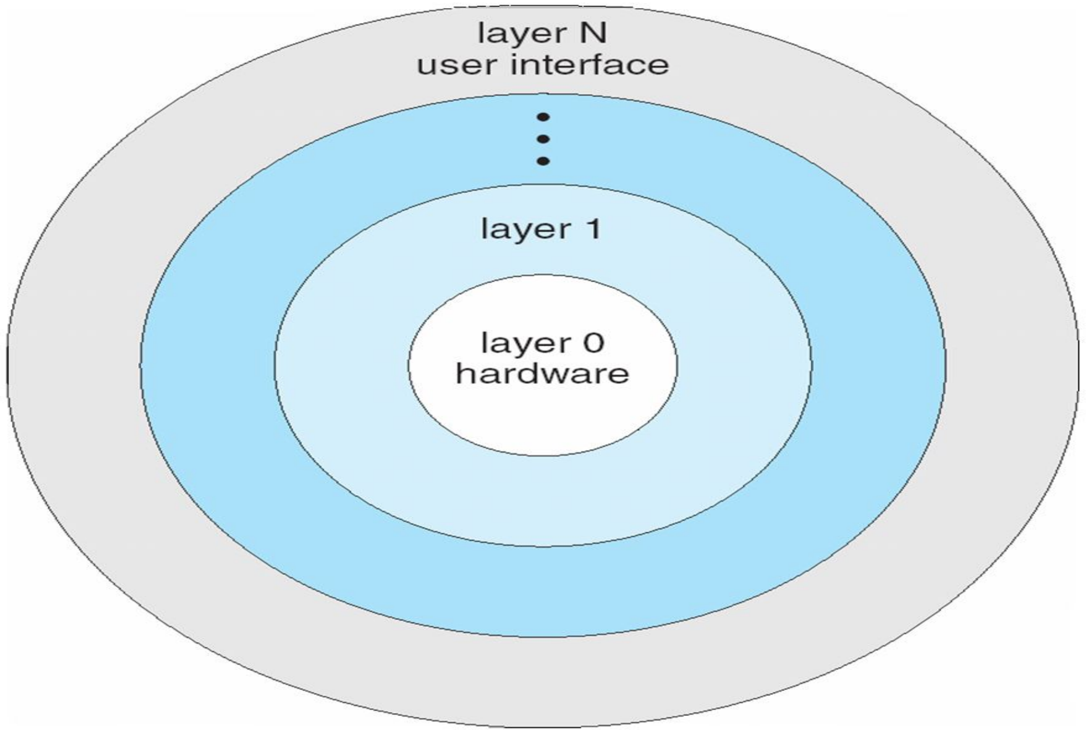
- Simple , small and limited system and grow beyond their original scope.

Disadvantages of simple structure

- In MS-DOS, the interfaces and levels of functionality are not well separated. For instance, application programs are able to access the basic *I/O routines* to write directly to the display and disk drives. Such freedom causes entire system to crash when user programs fail.
- No hardware protection and no dual mode operation.

2.7.2 Layered approach

- With proper hardware support, operating systems can be broken into pieces that are smaller and more appropriate than MS-DOS and UNIX.
- In layered approach, the operating system is broken into a number of layers(levels).
- The bottom layer is hardware (layer 0) and highest layer(layer N) is user interface. The following figure shows layered structure.
- An operating-system layer is an implementation of an abstract object made up **of data** and the **operations** that can manipulate those data.



Continued….

Advantages of layered approach

- Simple to construct and debug.
- The operating system can then retain much greater control over the computer and over the applications that make use of that computer.
- The layers are selected so that each **uses functions** (operations) and **services of** only lower-level layers. This approach simplifies debugging and system verification.
- Provides information hiding.

Disadvantages of layered approach

- The major difficulty with the layered approach involves appropriately defining the various layers. Because a layer can use only lower-level layers, careful planning is necessary.
- they tend to be less efficient than other types. For instance, when a user program executes an I/0 operation, it executes a system call that is trapped to the I/0 layer, which calls the memory-management layer which in turn calls the CPU-scheduling layer, which is then passed to the hardware.

2.7.3 Microkernels

- We have already seen that as UNIX expanded, the kernel became large and difficult to manage.
- University developed an operating system called Mach that modularized the kernel using the microkernel approach.
- This method structures the operating system by removing all nonessential components from the kernel and implementing them as system and user level programs. **The result is smaller kernel called microkernel**.
- Microkernel provides minimal process and memory management.
- **The main function of the micro kernel is to provide a communication facility between the client program and the various services that are also running in user space.**

Advantages:

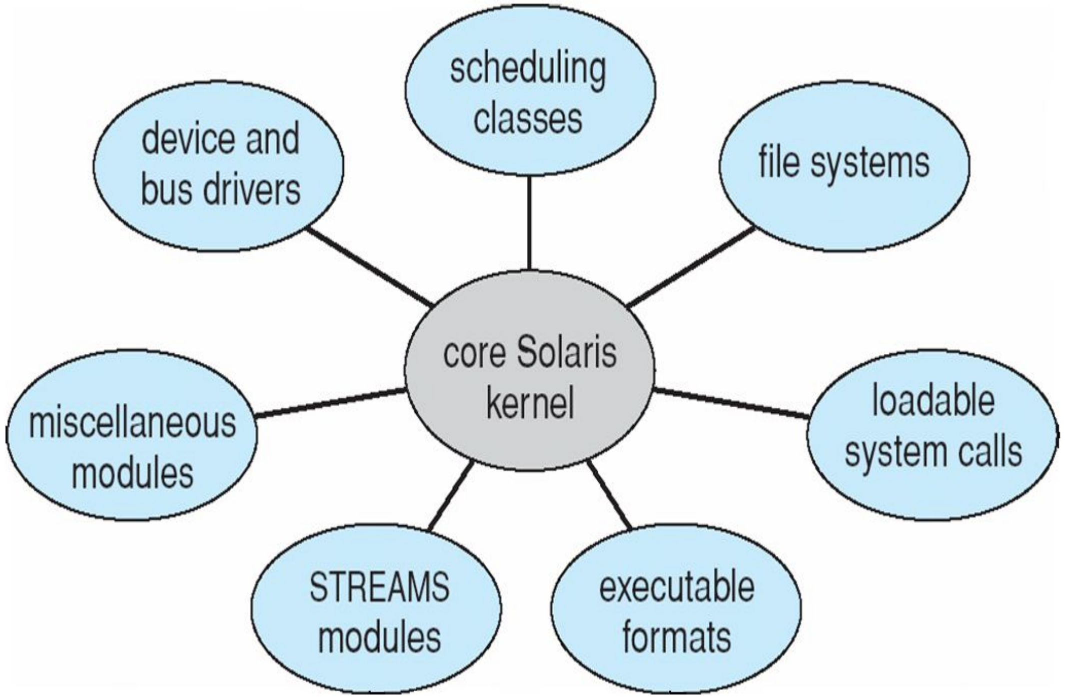
- Easy to extend operating system.
- Easy to port operating system from one hardware design to other.
- It provides more security and reliability.

Disadvantages

- decreased performance due to increased system function overhead.

2.7.4 Modules (modular approach)

- Perhaps the best current methodology for operating-system design involves using object-oriented programming techniques to create a modular kernel.
- Here, the kernel has a set of core components and links in additional services either during boot time or during run time.
- For example, the Solaris operating system structure, shown in Figure, is organized around a core kernel with seven types of loadable kernel modules:
- Scheduling classes
- File systems
- Loadable system calls
- Executable formats
- STREAMS modules
- Miscellaneous
- Device and bus drivers

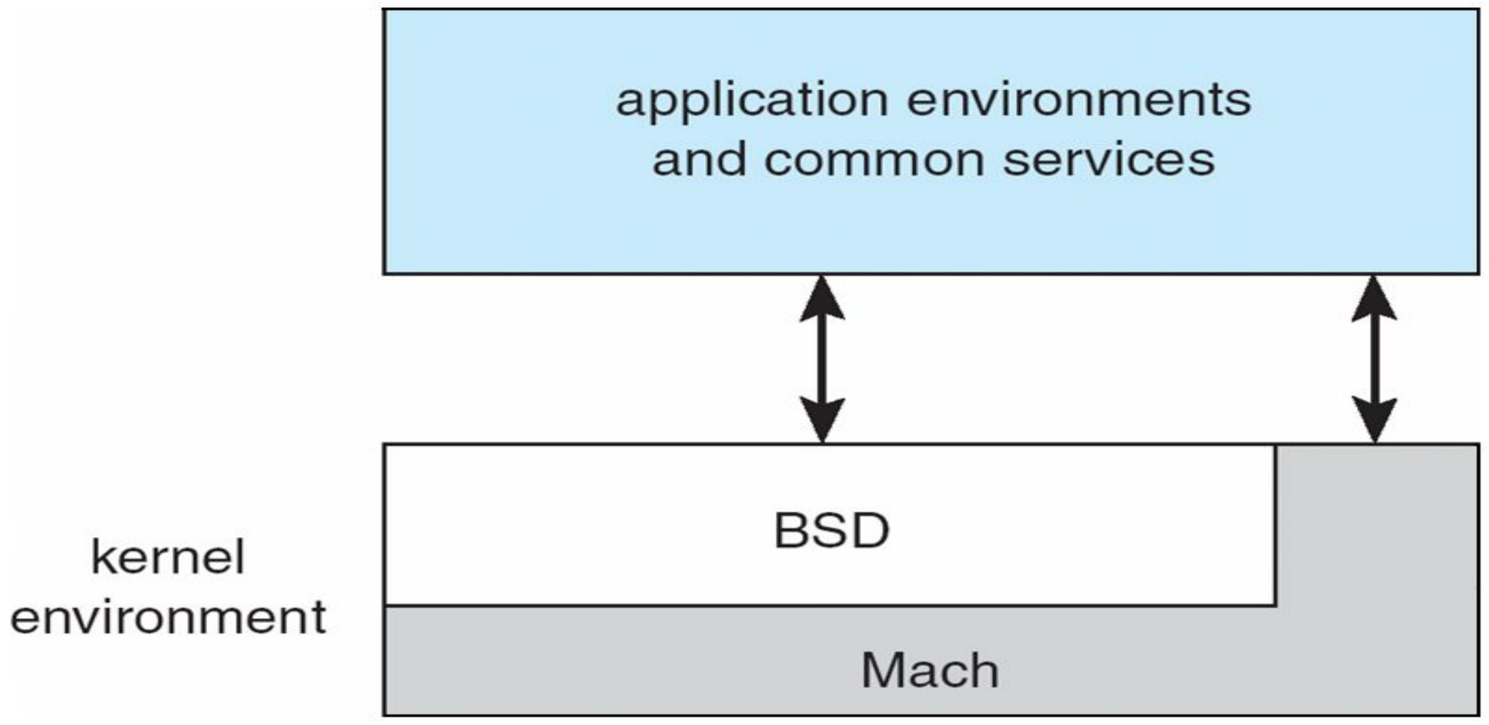


Continued….

- **The overall result resembles a layered system in that each kernel section has defined**, protected interfaces; but it is more flexible than a layered system in that any module can call any other module.
- **Furthermore, the approach is like the microkernel approach in that the primary module has only core functions and knowledge of how to load and communicate with other modules**; but it is more efficient, because modules do not need to invoke message passing in order to communicate.
- **The Apple Mac OS X** operating system in following figure uses a hybrid structure. It is a layered system in which one layer consists of the Mach microkernel.
- The top layers include application environments and a set of services providing a graphical interface to applications. Below these layers is the kernel environment, which consists primarily of the Mach microkernel and the BSD kernel.

Continued..

- Mach provides memory management; support for remote procedure calls (RPCs) and interprocess communication (IPC) facilities, including message passing.
- The BSD component provides a BSD command line interface, support for networking and file systems.



System Boot

- When power initialized on system, execution starts at a fixed memory location
  - Firmware ROM used to hold initial boot code
- Operating system must be made available to hardware so hardware can start it
  - Small piece of code – **bootstrap loader**, stored in **ROM** or **EEPROM** locates the kernel, loads it into memory, and starts it
  - Sometimes two-step process where **boot block** at fixed location loaded by ROM code, which loads bootstrap loader from disk
- Common bootstrap loader, **GRUB**, allows selection of kernel from multiple disks, versions, kernel options
- Kernel loads and system is then **running**