SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited with Grade "A" by NAAC | Approved by AICTE

SCS1301 - OPERATING SYSTEM

UNIT – 1

INTRODUCTION TO OS

Dr.R.Subhashini
Professor and Head
Dept. of Information Technology
School of Computing

SATHYABAMA UNIVERSITY **FACULTY OF COMPUTING**

SCS1301	OPERATING SYSTEM	L	T	P	Credits	Total Marks
		3	0	0	3	100

COURSE OBJECTIVES

- To have an overview of different types of operating systems.
- To understand the concept of process management.
- To understand the concept of storage management.
- To understand the concept of I/O and file systems.

UNIT 1 INTRODUCTION 8 Hrs.

Introduction - Operating system structures - System components - OS services - System calls - System structure - Resources Processes - Threads - Objects - Device management - Different approaches - Buffering device drivers.

UNIT 2 PROCESS MANAGEMENT 9 Hrs.

Processes - Process concepts - Process scheduling - Operations on processes - Cooperating processes - CPU scheduling - Basic concepts - Scheduling criteria - Scheduling algorithms - Preemptive strategies - Non-preemptive strategies

UNIT 3 SYNCHRONIZATION AND DEADLOCKS 9 Hrs

The critical section problem - Semaphores - Classic problems of synchronization - Critical regions - Monitors Deadlocks - Deadlock characterization - Prevention - Avoidance - Detection - Recovery.

UNIT 4 MEMORY MANAGEMENT 9 Hrs

Storage Management Strategies - Contiguous Vs. Non-Contiguous Storage Allocation - Fixed & Variable Partition Multiprogramming - Paging - Segmentation - Paging/Segmentation Systems - Page Replacement Strategies - Demand & Anticipatory Paging - File Concept - Access Methods - Directory Structure - File Sharing - Protection File - System Structure - Implementation.

UNIT 5 I/O SYSTEM,LINUX & SHELL PROGRAMMING **10 Hrs.**

Mass Storage Structure - Disk Structure- Disk Scheduling - Disk Management - Swap Space Management - RAID Structure - Shell Operation Commends - Linux File Structure - File Management Operation - Internet Service - Telnet - FTP - Filters & Regular Expressions - Shell Programming - Variable, Arithmetic Operations, Control Structures, Handling Date, Time & System Information.

Max. 45 Hours

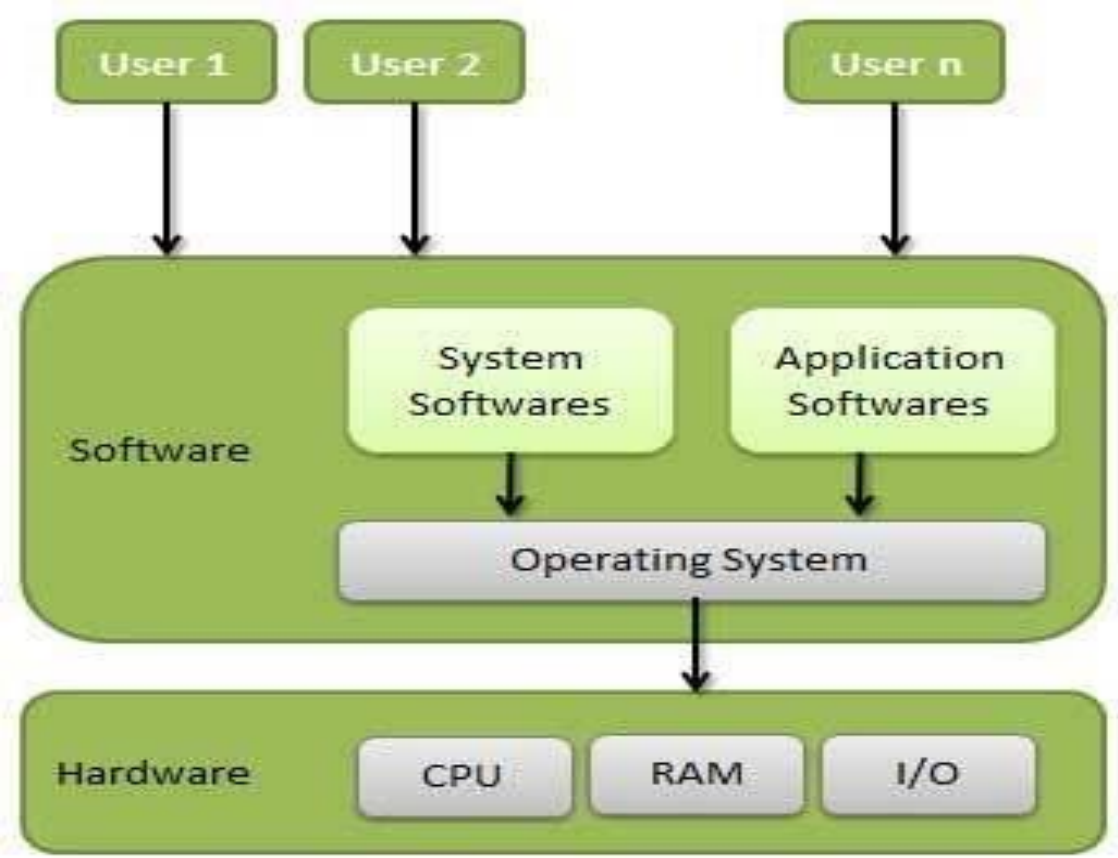
TEXT / REFERENCE BOOKS

1. Abraham Silberschatz, Peter Galvin and Gagne, "Operating System Concepts", 6th Edition, Addison Wesley, 2002.
2. Harvey M.Deitel, "Operating System", 2nd Edition, Addison Wesley, 2000.
3 Gary Nutt, "Operating System, A modern perspective", 2nd Edition, Addison Wesley, 2000.
4 Richard Peterson, "Linux : The Complete Reference", 6th Edition, Tata McGraw Hills, 2007.

INTRODUCTION TO OS

- A computer system has many resources (hardware and software), which may be require to complete a task.
- Operating System is a system software that acts as an intermediary between a user and Computer Hardware to enable convenient usage of the system and efficient utilization of resources.
- The commonly required resources are input/output devices, memory, file storage space, CPU etc.
- The operating system acts as a manager of the above resources and allocates them to specific programs and users, whenever necessary to perform a particular task.
- The resources are processor, memory, files, and I/O devices. In simple terms, an operating system is the interface between the user and the machine.
- Therefore operating system is the resource manager i.e. it can manage the resource of a computer system internally.

INTRODUCTION TO OS



GOALS OF OPERATING SYSTEM

- Execute user programs
- Make solving user problems easier.
- Make the computer system convenient to use.
- Use the computer hardware in an efficient manner.

FUNCTIONS OF OPERATING SYSTEM / SYSTEM COMPONENTS

- Main Memory Management
- Processor Management
- Device Management
- File Management
- I/O System Management
- Secondary Management
- Networking
- Protection System
- Command Interpreter System

MEMORY MANAGEMENT

- Memory management refers to management of Primary Memory or Main Memory. Main memory is a large array of words or bytes where each word or byte has its own address.
- Main memory provides a fast storage that can be accessed directly by the CPU.
- **ACTIVITIES**
  - Keeps tracks of primary memory, i.e., what part of it are in use by whom, what part is not in use.
  - In multiprogramming, the OS decides which process will get memory when and how much.
  - Allocates the memory when a process requests it to do so.
  - De-allocates the memory when a process no longer needs it or has been terminated

PROCESSOR MANAGEMENT

- In multiprogramming environment, the OS decides which process gets the processor when and for how much time. This function is called process scheduling
- **ACTIVITIES**
  - Keeps tracks of processor and status of process. The program responsible for this task is known as traffic controller.
  - Allocates the processor (CPU) to a process.
  - De-allocates processor when a process is no longer required.

DEVICE MANAGEMENT

- An Operating System manages device communication via their respective drivers
- **ACTIVITIES**
  - Keeps tracks of all devices. Program responsible for this task is known as the I/O controller.
  - Decides which process gets the device when and for how much time.
  - Allocates the device in the efficient way.
  - De-allocates devices.

FILE MANAGEMENT

- A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions.
- **ACTIVITIES**
  - Keeps track of information, location, uses, status etc. The collective facilities are often known as file system.
  - Decides who gets the resources.
  - Allocates the resources.
  - De-allocates the resources

I/O SYSTEM MANAGEMENT

- OS hides the peculiarities of specific hardware devices from the user.
- **ACTIVITIES**
  - A general device-driver interface
  - Drivers for specific hardware devices.
  - Only the device driver knows the peculiarities of the specific device to which it is assigned.

SECONDARY STORAGE MANAGEMENT

- The main purpose of a computer system is to execute programs. These programs, with the data they access, must be in main memory, or primary storage.
- Systems have several levels of storage, including primary storage, secondary storage and cache storage.
- **ACTIVITIES**
  - Free-space management (paging/swapping)
  - Storage allocation (what data goes where on the disk)
  - Disk scheduling (Scheduling the requests for memory access).

NETWORKING

- A distributed systems are a collection of processors that do **not share memory, peripheral devices,** or a clock.
- The processors in the system are connected through a communication-network. Communication takes place using a protocol. The network may be fully or partially connected
- The communication-network design must consider routing and connection strategies.
- **ACTIVITIES**
  - Computation Speed-up
  - Increased functionality
  - Increased data availability
  - Enhanced reliability

PROTECTION SYSTEM

- If a computer system has multiple users and allows the concurrent execution of multiple processes, then the various processes must be protected from one another's activities.
- Protection refers to mechanism for controlling the access of programs, files, memory segments, processes (CPU) only by the users who have gained proper authorization from the OS.
- **ACTIVITIES**
  - Distinguish between authorized and unauthorized usage.
  - Specify the controls to be imposed.
  - Provide a means of enforcement.

COMMAND INTERPRETER SYSTEM

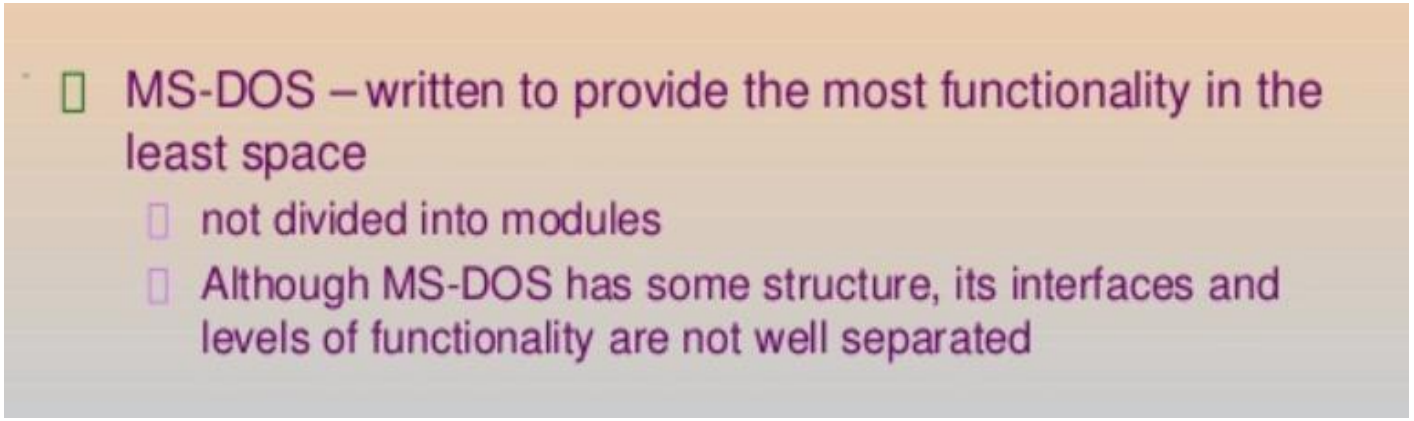
- A command interpreter is one of the important system programs for an OS.
- It is an interface of the operating system with the user. The user gives commands, which are executed by Operating system (usually by turning them into system calls).
- The main function of a command interpreter is to get and execute the next user specified command.
- **ACTIVITIES**
  - process creation and management
  - I/O handling
  - secondary-storage management
  - main-memory management
  - file-system access
  - protection
  - networking

CLASSIFICATION OF OPERATING SYSTEM

- **Multi-user OS** : Allows two or more users to run programs at the same time.
- **Multiprocessing OS** : Support a program to run on more than one central processing unit (CPU) at a time.
- **Multitasking OS** : Allows to run more than one program at a time
- **Multithreading OS** : Allows different parts of a single program to run concurrently (simultaneously or at the same time).
- **Real time OS** : These are designed to allow computers to process and respond to input instantly

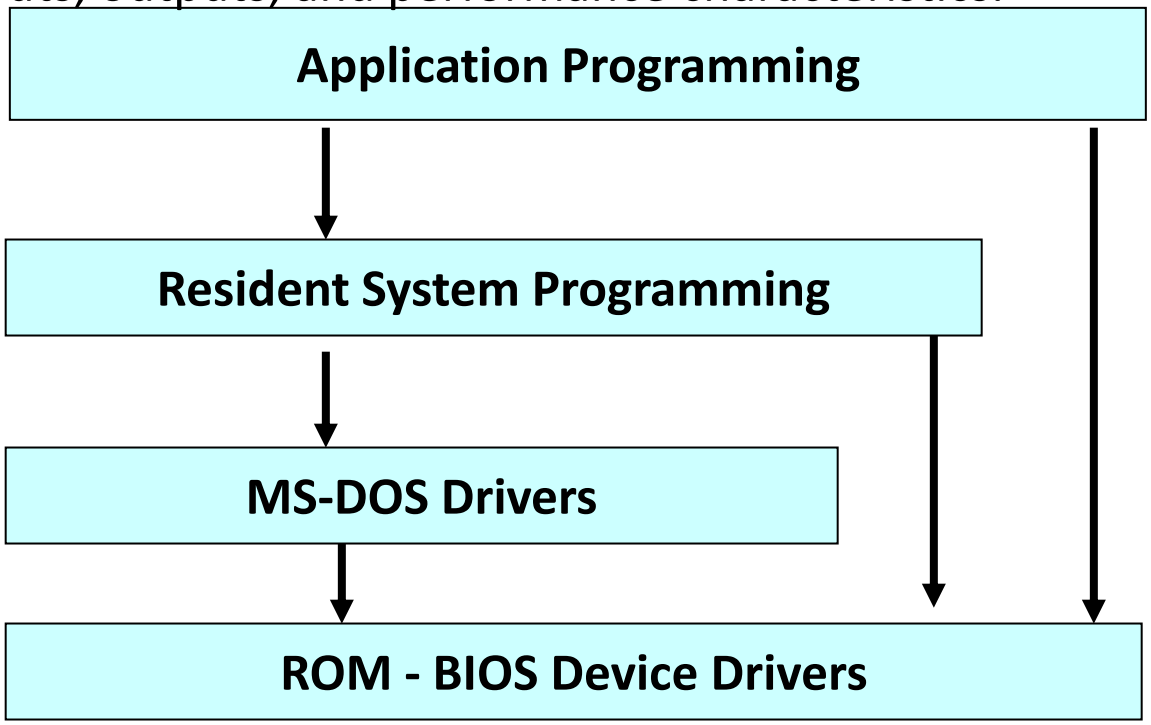
OPERATING SYSTEM STRUCTURES

- **A SIMPLE STRUCTURE:** There are many operating systems that have a rather simple structure.
- These started as small systems and rapidly expanded much further than their scope. A common example of this is MS-DOS.
- **MS – DOS SYSTEM STRUCTURE**



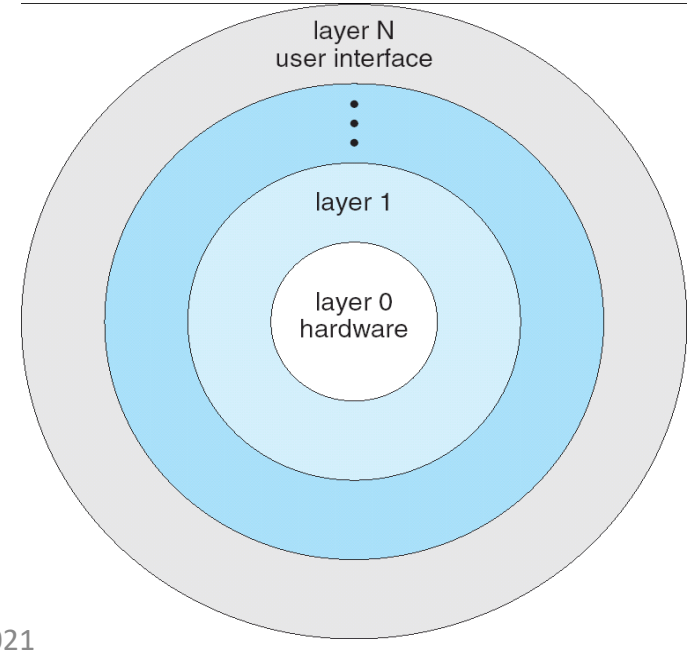
MS – DOS Structure

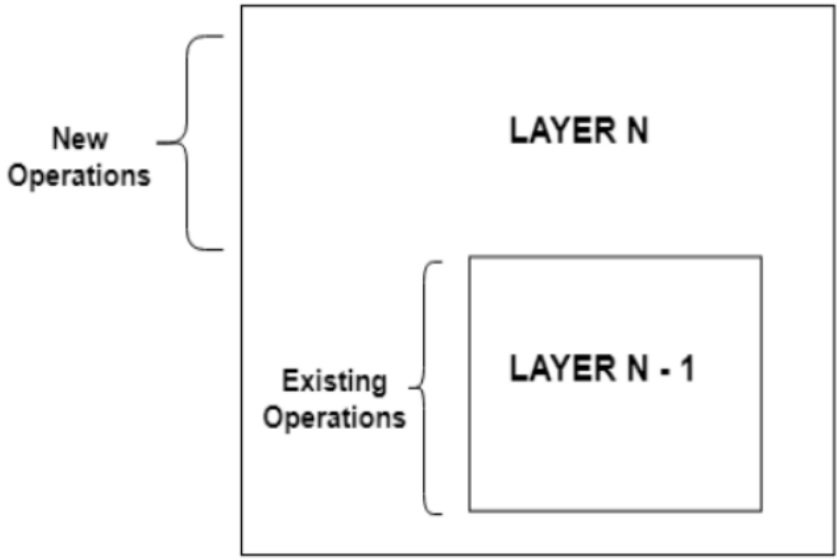
For efficient performance and implementation an OS should be partitioned into separate subsystems, each with carefully defined tasks, inputs, outputs, and performance characteristics.



LAYERED STRUCTURE

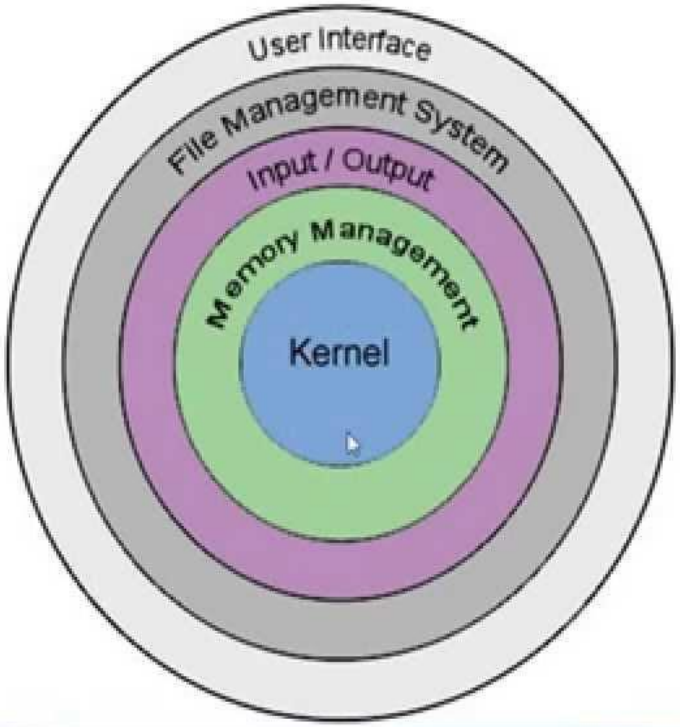
- One way to achieve modularity in the operating system is the layered approach. It breaks the OS into a number of smaller layers, each of which rests on the layer below it, and relies solely on the services provided by the next lower layer.





Layered Structure of Operating System

Layered Structure

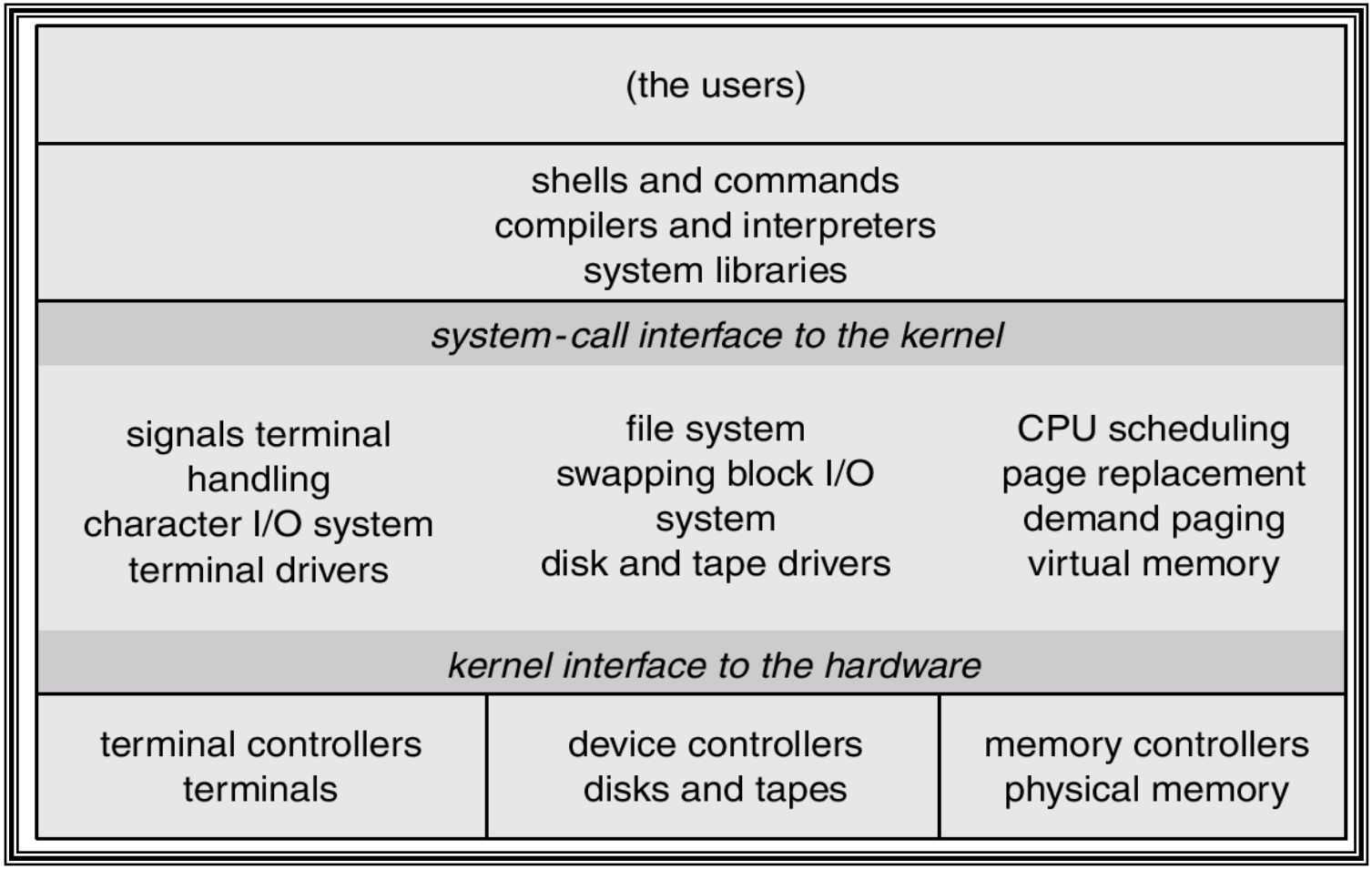


UNIX STRUCTURE

UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts.

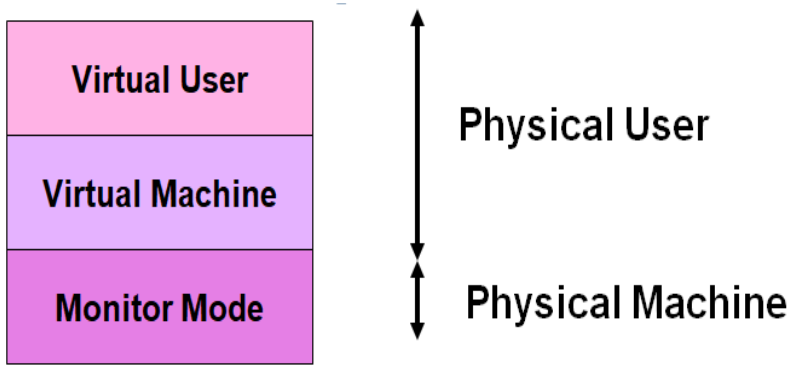
- Systems programs
- The kernel
  - Consists of everything below the system-call interface and above the physical hardware
  - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level.

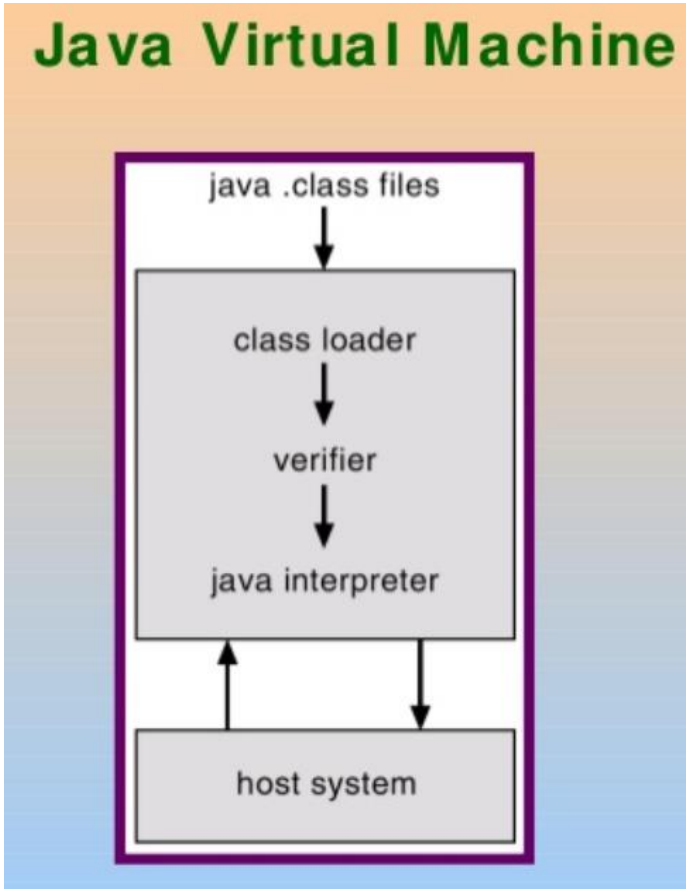
UNIX


(the users)
shells and commands
compilers and interpreters
system libraries
system-call interface to the kernel
signals terminal handling
character I/O system
terminal drivers
file system
swapping block I/O system
disk and tape drivers
CPU scheduling
page replacement
demand paging
virtual memory
kernel interface to the hardware
terminal controllers
terminals
device controllers
disks and tapes
memory controllers
physical memory

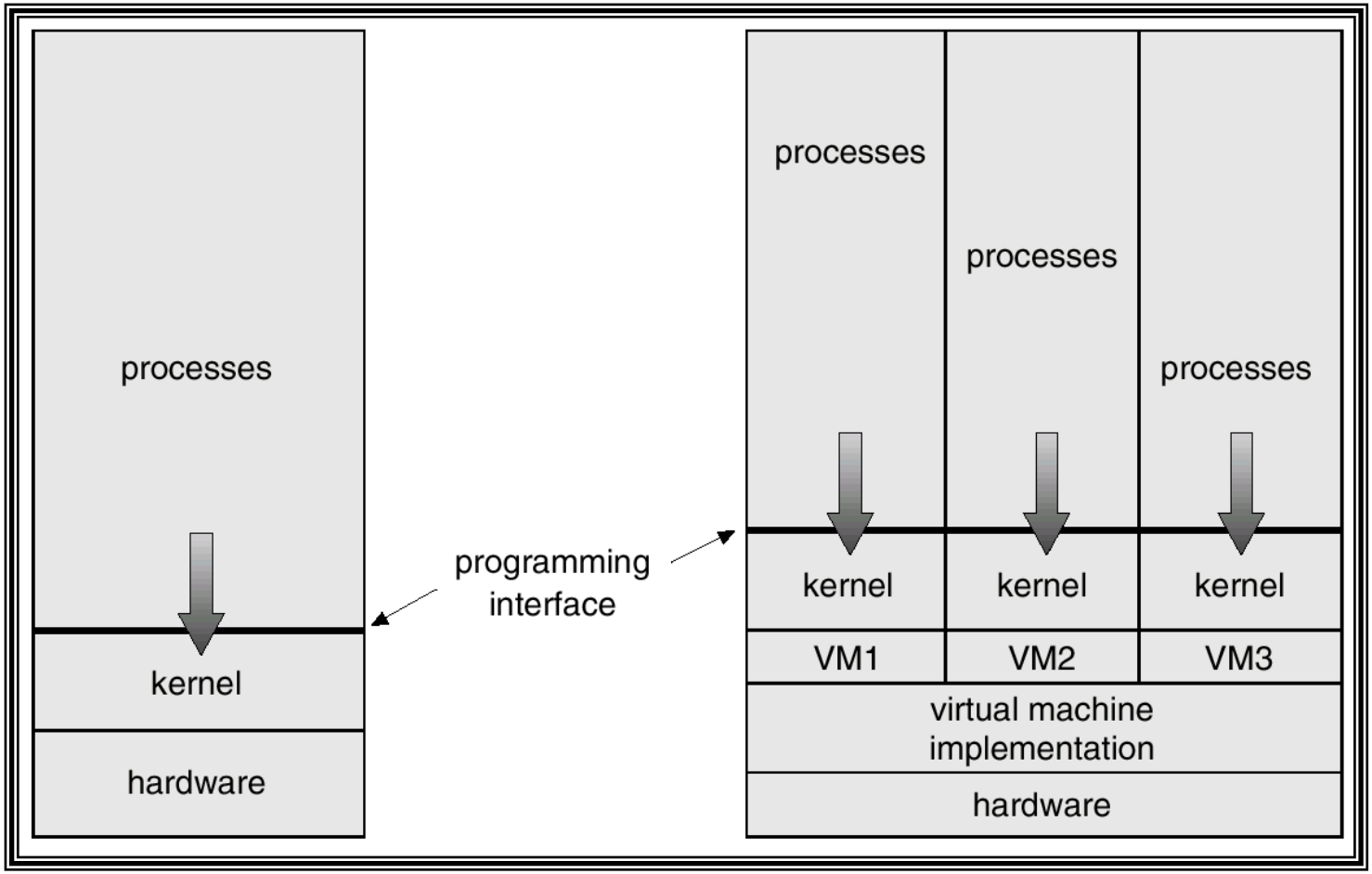
VIRTUAL MACHINE

- In a Virtual Machine - each process "seems" to execute on its own processor with its own memory, devices, etc.
- The resources of the physical machine are shared. Virtual devices are sliced out of the physical ones. Virtual disks are subsets of physical ones.





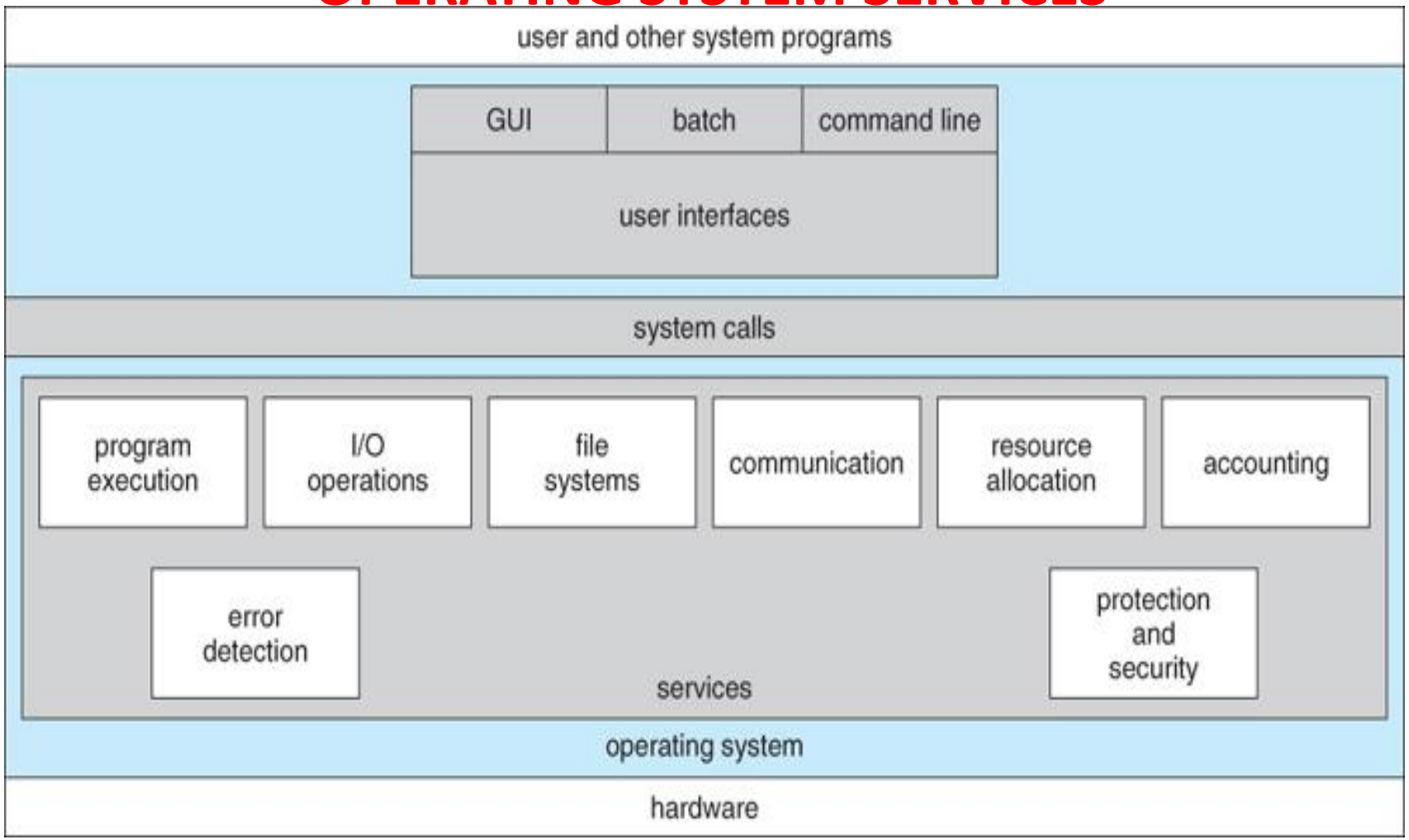
VIRTUAL MACHINE



OPERATING SYSTEM SERVICES

- An OS provides environment for the execution of programs.
- It provides certain services to programs and to the users of those programs.
- These OS services are provided for the convenience of the programmer, to make the programming task easier. One set of operating-system services provides functions that are helpful to the user are,
  - **PROGRAM EXECUTION**
  - **I/O OPERATIONS**
  - **FILE-SYSTEM MANIPULATION**
  - **COMMUNICATIONS**
  - **ERROR DETECTION**
  - **RESOURCE ALLOCATION**
  - **PROTECTION AND SECURITY**

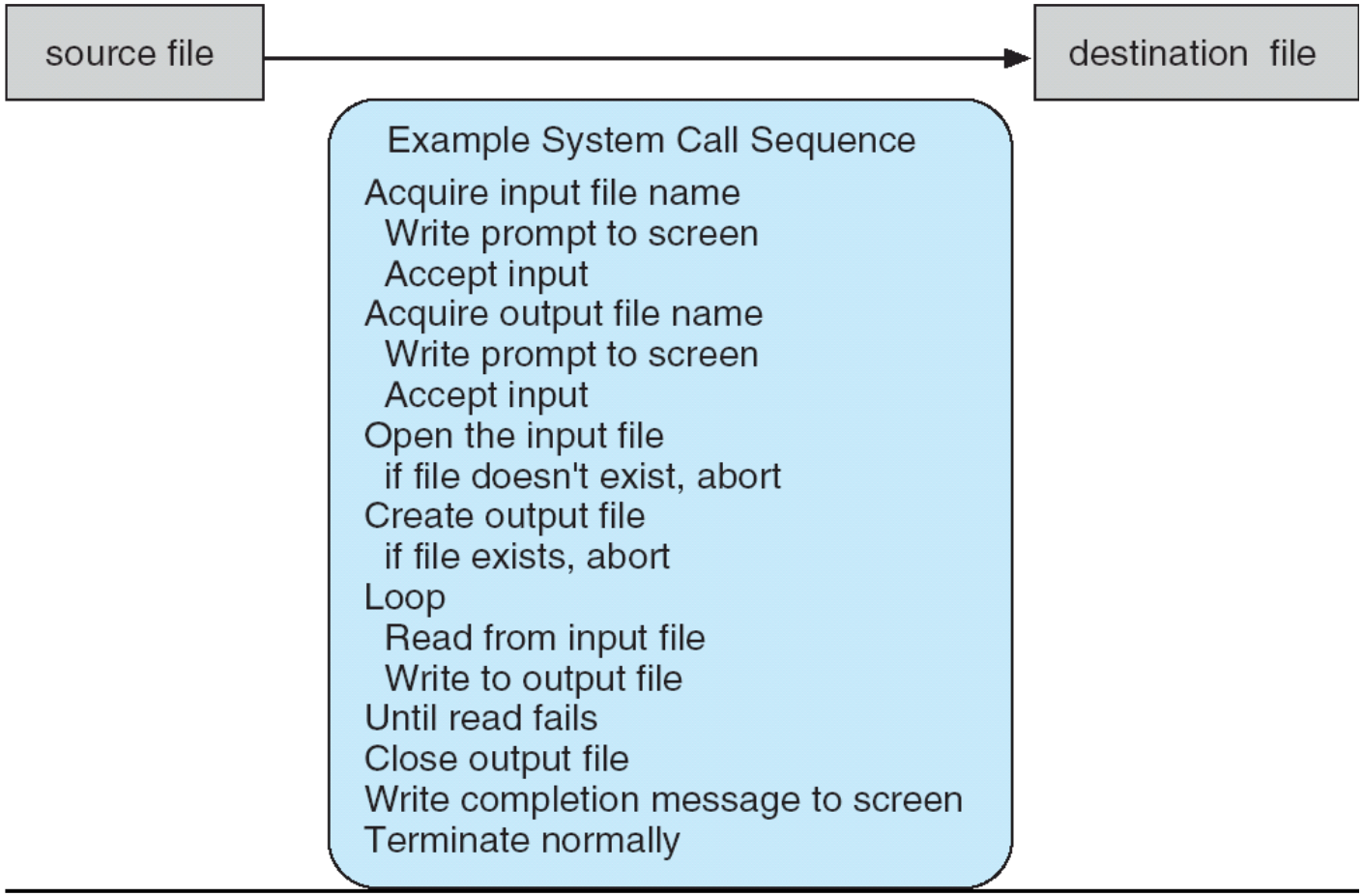
OPERATING SYSTEM SERVICES



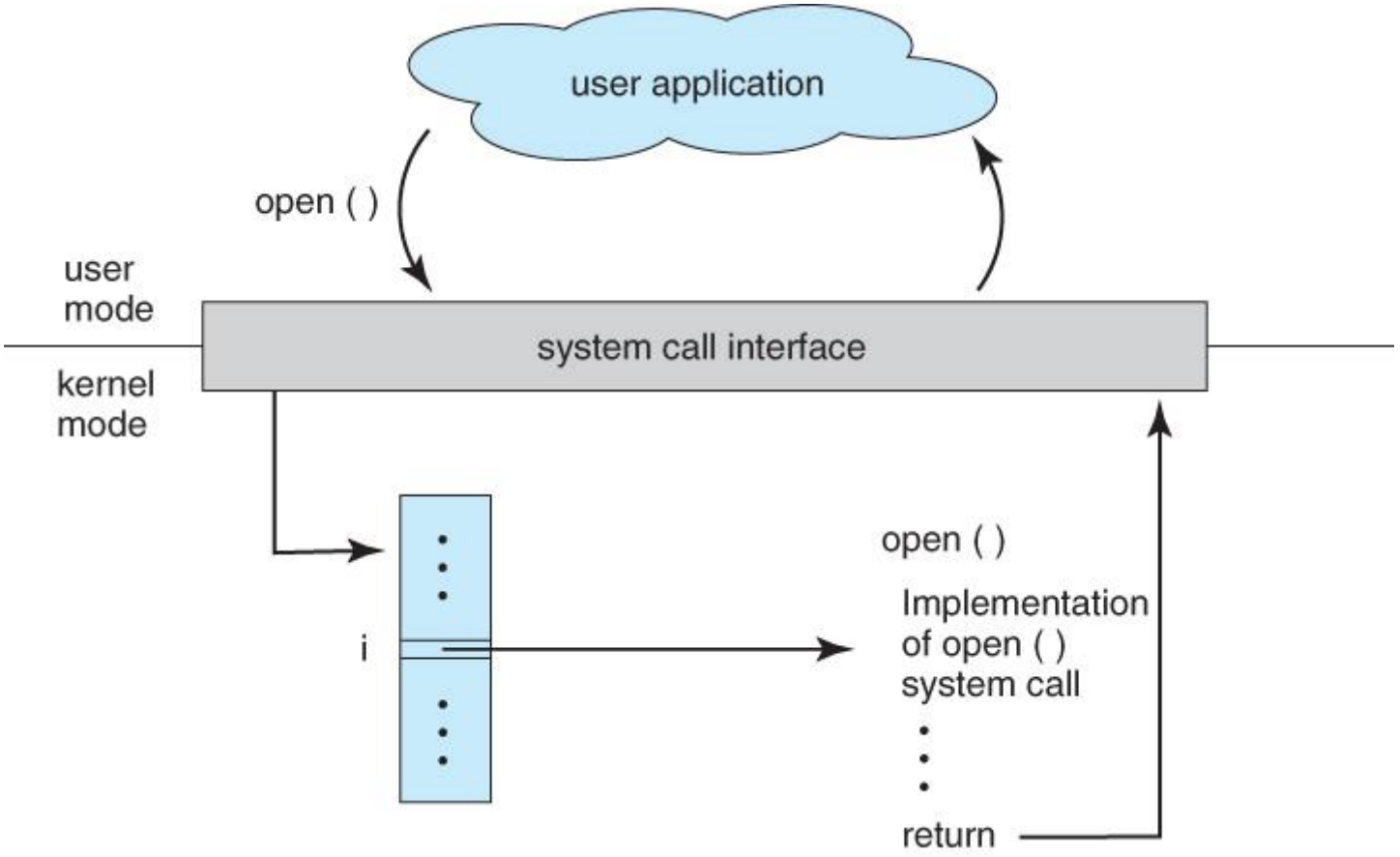
SYSTEM CALLS

- System calls provide the interface between a process and the operating system.
- 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)

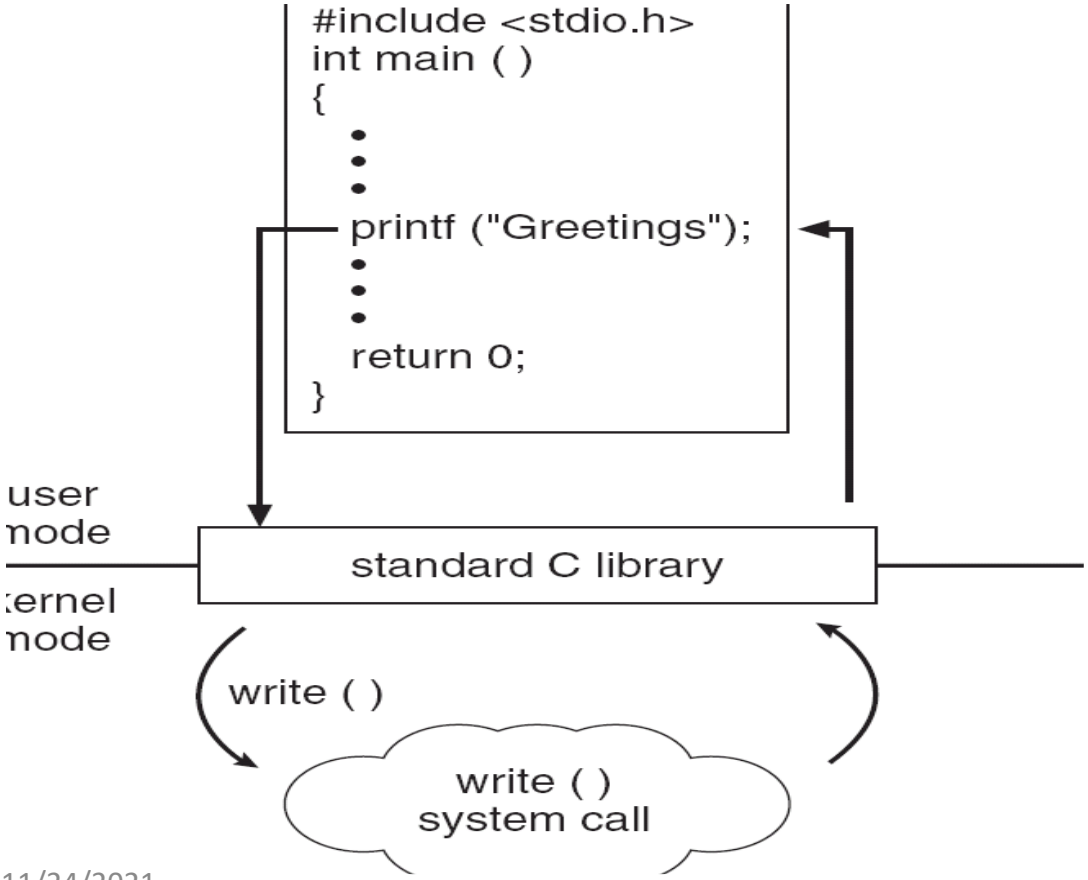
Illustrates the sequence of system calls required to copy a file



User application invoking the open( ) system call



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

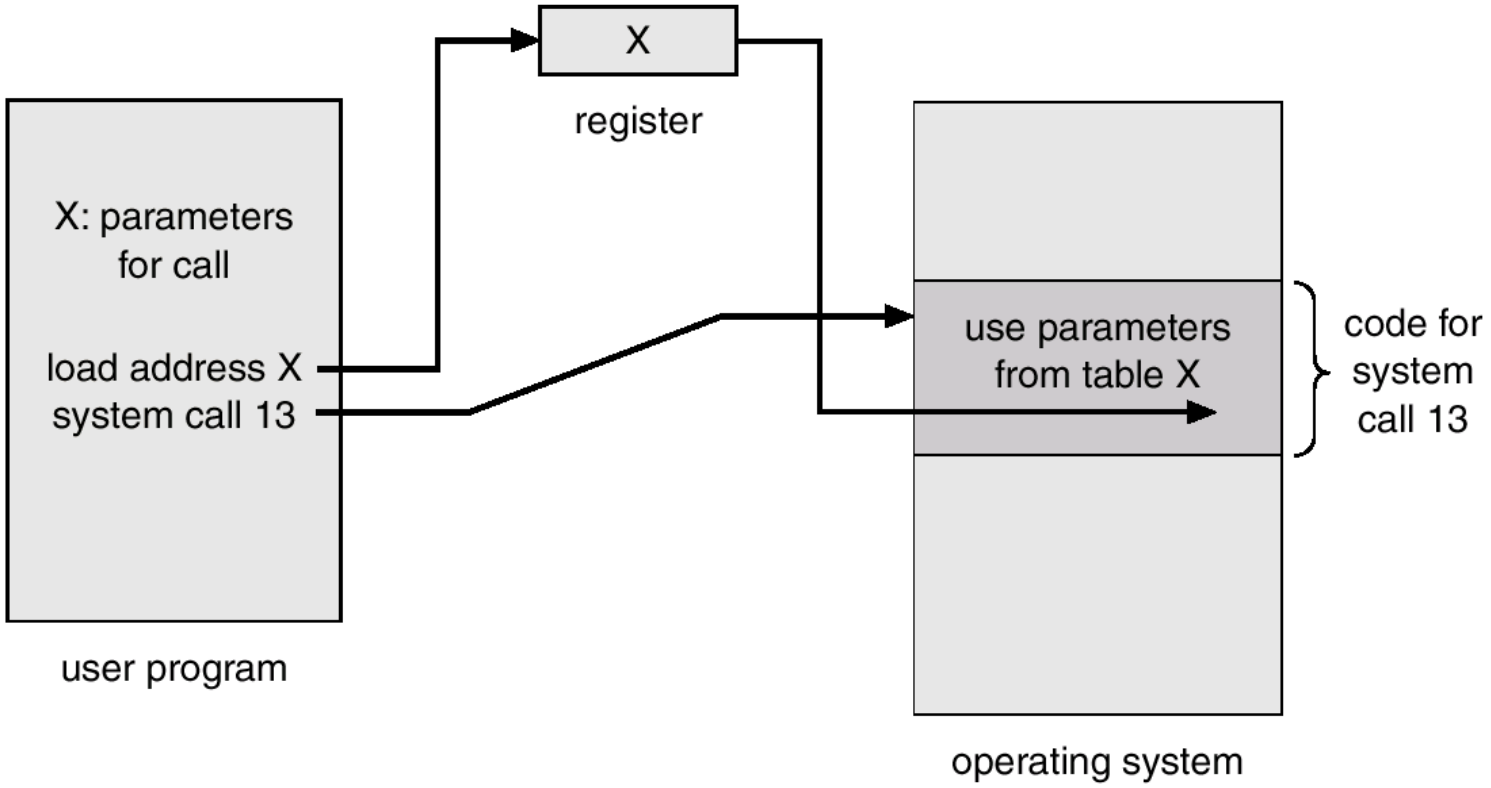


Standard C Library Example

Three general methods are used to pass parameters between a running program and the operating system.

- Simplest approach is to pass parameters in registers.
- Store the parameters in a table in memory, and the table address is passed as a parameter in a register
- Push (store) the parameters onto the stack by the program, and pop off the stack by operating system.

Passing of parameters as a table



TYPES OF SYSTEM CALLS

- System calls can be grouped roughly in to five categories:
  - **Process control**
  - **File management**
  - **Device management**
  - **Information maintenance**
  - **Communications**

Types of System Calls

- 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

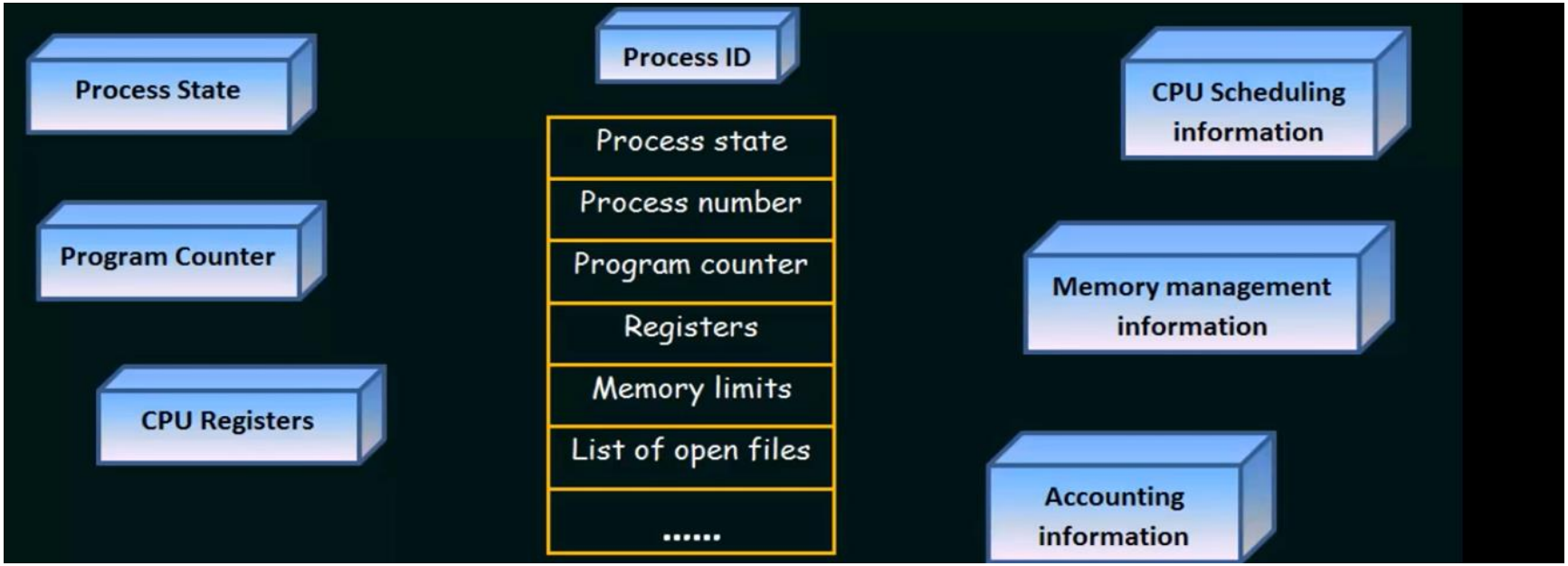
Examples of Windows and Unix System Calls

	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()

Processes

- A process can be thought of as a **program in execution**, A process will need certain resources
  - such as **CPU time, memory, files, and I/O devices**
  - to accomplish its task.
- Process is an Active Entity and Program is a Passive Entity.
- A process is the unit of work in most systems.
- Systems consist of a collection of processes:
  - Operating-system processes execute **system code**, and
  - user processes execute **user code**.
  - All these processes may execute concurrently.

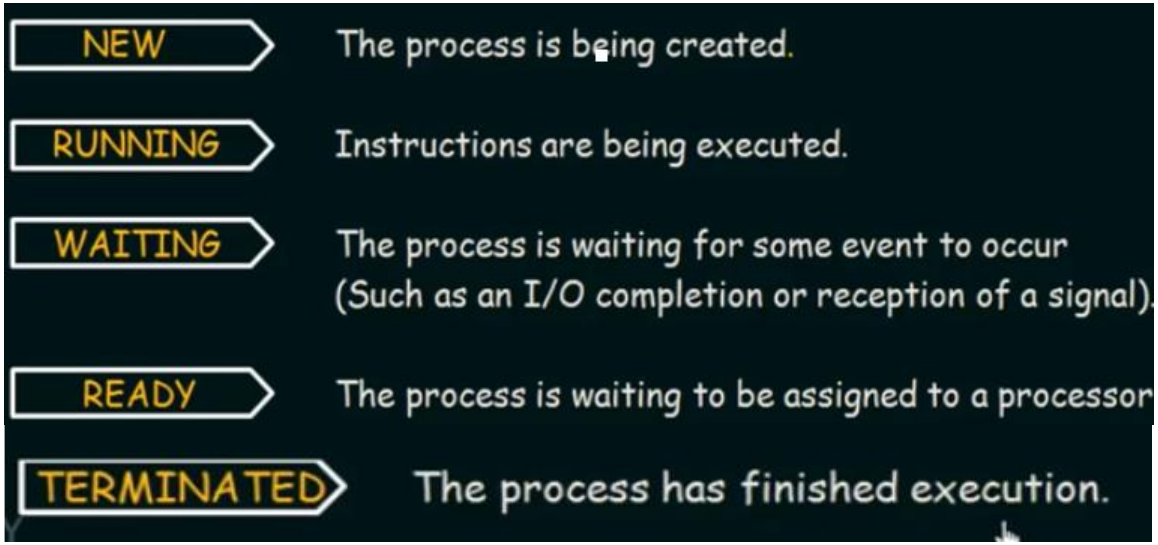
Process Control Block(data structure used by computer operating systems to store all the information about a process)



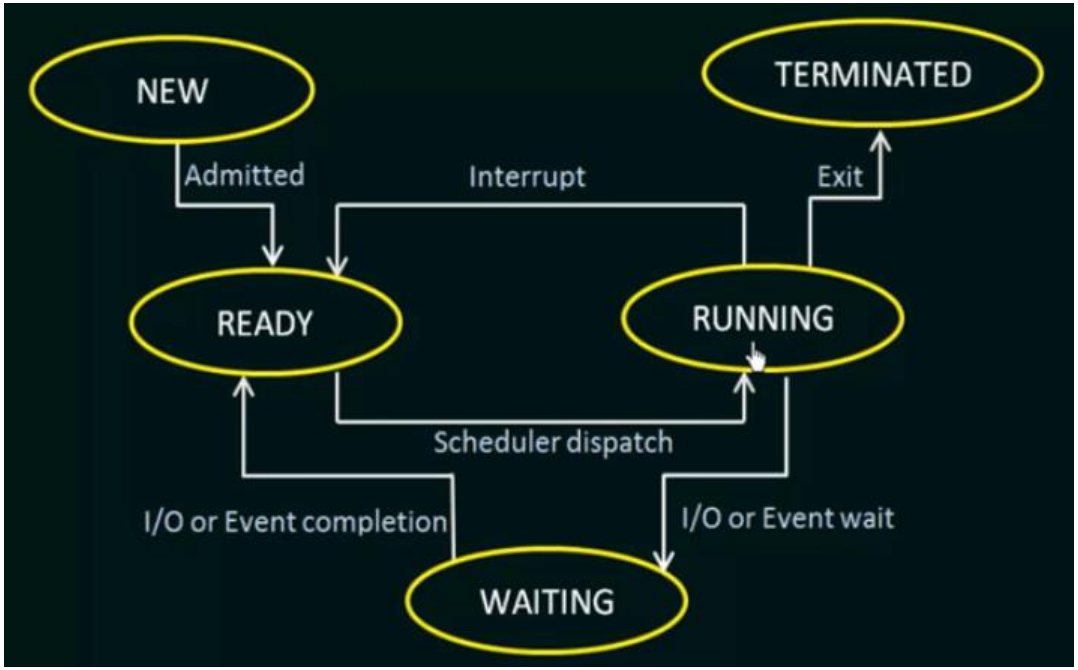
Process Control Block

- **Process ID** (PID)–unique identification number for each process
- **Process State**–New, Ready, Running, Waiting, Terminated
- **Program Counter** (PC)–A pointer to the address of the next instruction to be executed for this process
- **List of Open files**– This information includes the list of files opened for a process.
- **Memory limits** – This field contains the information about memory management system used by operating system.
- **CPU Registers**–Register set where process needs to be stored for execution for running state eg: accumulator, base and general purpose registers.

Process States



Process States



Resources

The OS treats an **entity as a resource** if it satisfies the below characteristics:

- A process must request it from the OS.
- A process must suspend its operation until the entity is allocated to it.

The most common source is a **file**. A process must request a file before it can read it or write it. Further, if the file is unavailable, the process must wait until it becomes available. This abstract description of a resource is crucial to the way various entities(such as files, memory and devices) are managed.

Thread

A thread is a basic unit of **CPU utilization**. A thread, sometimes called as **light weight process** whereas a process is a **heavyweight process**.

- Thread comprises:
  - A thread ID
  - A program counter
  - A register set
  - A stack.
- A process is a program that performs a single thread of execution i.e., a process is a executing program with a single thread of control. For example, when a process is running a word- processor program, a single thread of instructions is being executed.

Thread

- The single thread of control allows the process to perform only one task at one time. For example, the user cannot simultaneously type in characters and run the spell checker within the same process.
- Traditionally a process contained only a single *thread* of control as it ran, many modern operating systems have extended the process concept to allow a process to have multiple threads of execution and thus to perform more than one task at a time.
- For example, in a browser, multiple tabs can be different threads

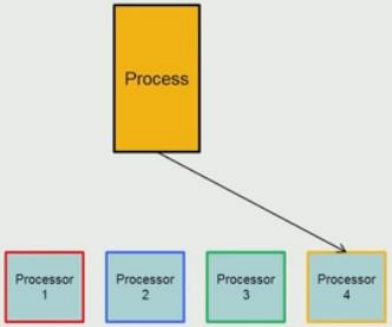
Thread

- The operating system is responsible for the following activities in connection with process and thread management:
  - The creation and deletion of both **user and system processes**
  - The **scheduling** of processes
  - The provision of mechanisms for **synchronization**
  - Communication
  - **Deadlock** handling for processes.

Process Model

Consider this Example

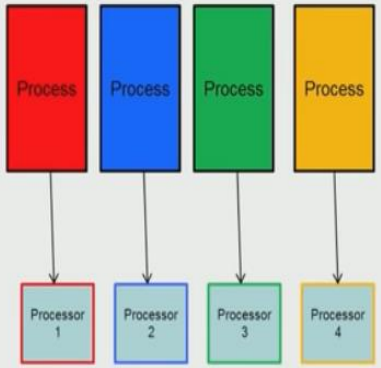
```c
#include <stdio.h>


unsigned long addall(){
  int i=0;
  unsigned long sum=0;

  while (i< 10000000){
     sum += i;
     i++;
  }
  return sum;
}


int main()
{
   unsigned long sum;
   srandom(time(NULL));
   sum = addall();
   printf("%lu\n", sum);
}
```



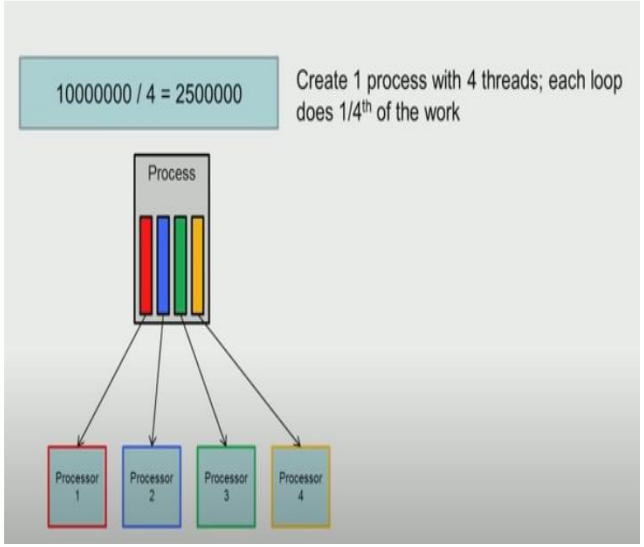
Speeding up with multiple processes

10000000 / 4 = 2500000

Create 4 processes, each loop does 1/4th of the work



Thread Model

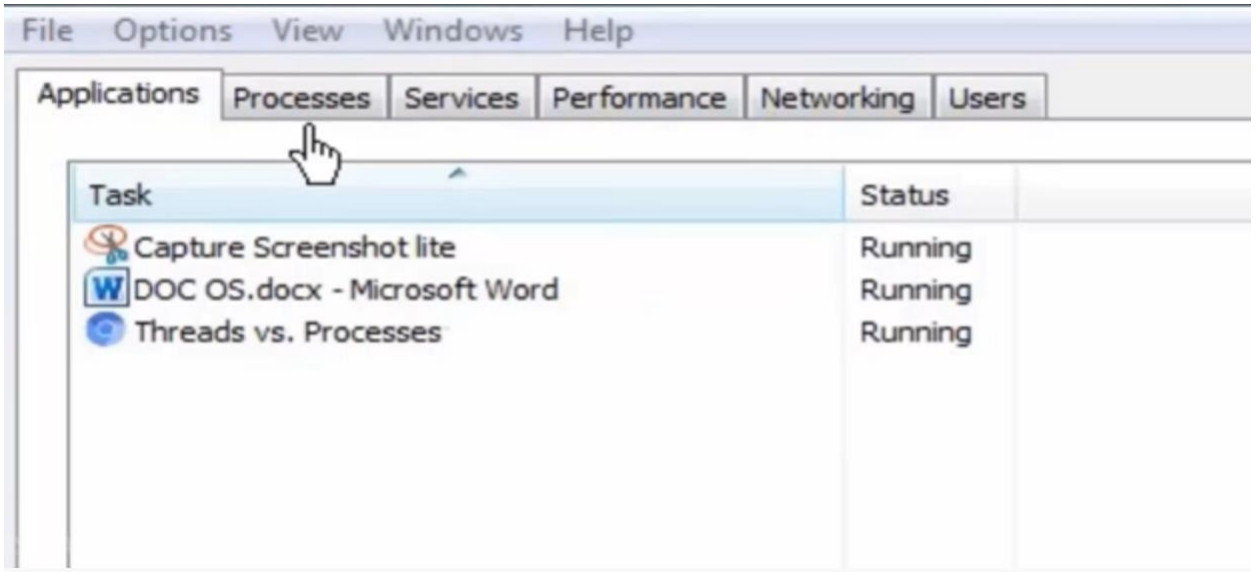



```c
#include <pthread.h>
#include <stdio.h>

unsigned long sum[4];

void *thread_fn(void *arg){
  long id = (long) arg;
  int start = id * 2500000;
  int i=0;

  while(i < 2500000){
     sum[id] += (i + start);
     i++;
  }
  return NULL;
}

int main(){
  pthread_t t1, t2, t3, t4;

  pthread_create(&t1, NULL, thread_fn, (void *)0);
  pthread_create(&t2, NULL, thread_fn, (void *)1);
  pthread_create(&t3, NULL, thread_fn, (void *)2);
  pthread_create(&t4, NULL, thread_fn, (void *)3);
  pthread_join(t1, NULL);
  pthread_join(t2, NULL);
  pthread_join(t3, NULL);
  pthread_join(t4, NULL);
  printf("%lu\n", sum[0] + sum[1] + sum[2] + sum[3]);
  return 0;
}
```

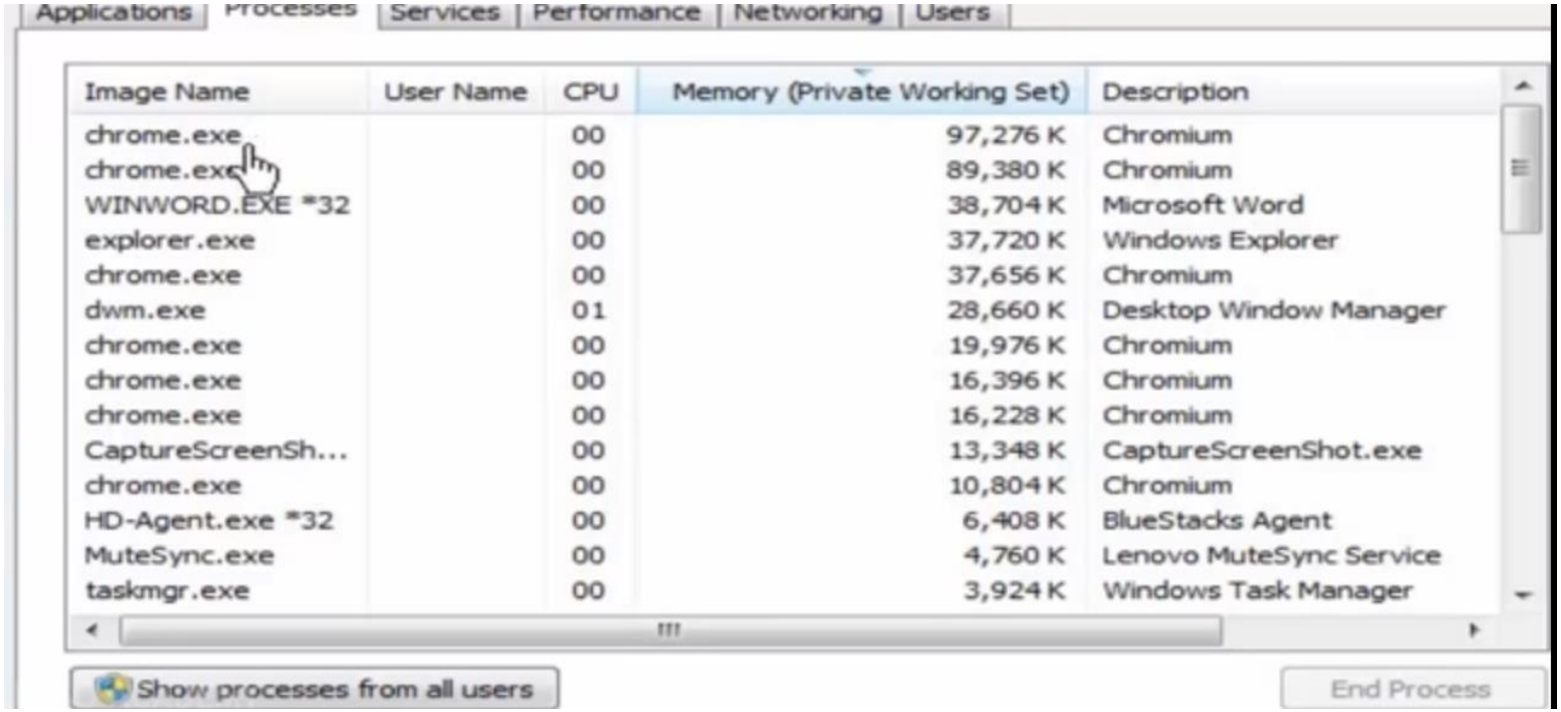
Creation time for 50,000 Process and Thread

Platform	fork()			pthread_create()		
	real	user	sys	real	user	sys
Intel 2.6 GHz Xeon E5-2670 (16 cores/node)	8.1	0.1	2.9	0.9	0.2	0.3
Intel 2.8 GHz Xeon 5660 (12 cores/node)	4.4	0.4	4.3	0.7	0.2	0.5
AMD 2.3 GHz Opteron (16 cores/node)	12.5	1.0	12.5	1.2	0.2	1.3
AMD 2.4 GHz Opteron (8 cores/node)	17.6	2.2	15.7	1.4	0.3	1.3
IBM 4.0 GHz POWER6 (8 cpus/node)	9.5	0.6	8.8	1.6	0.1	0.4
IBM 1.9 GHz POWER5 p5-575 (8 cpus/node)	64.2	30.7	27.6	1.7	0.6	1.1
IBM 1.5 GHz POWER4 (8 cpus/node)	104.5	48.6	47.2	2.1	1.0	1.5
INTEL 2.4 GHz Xeon (2 cpus/node)	54.9	1.5	20.8	1.6	0.7	0.9
INTEL 1.4 GHz Itanium2 (4 cpus/node)	54.5	1.1	22.2	2.0	1.2	0.6

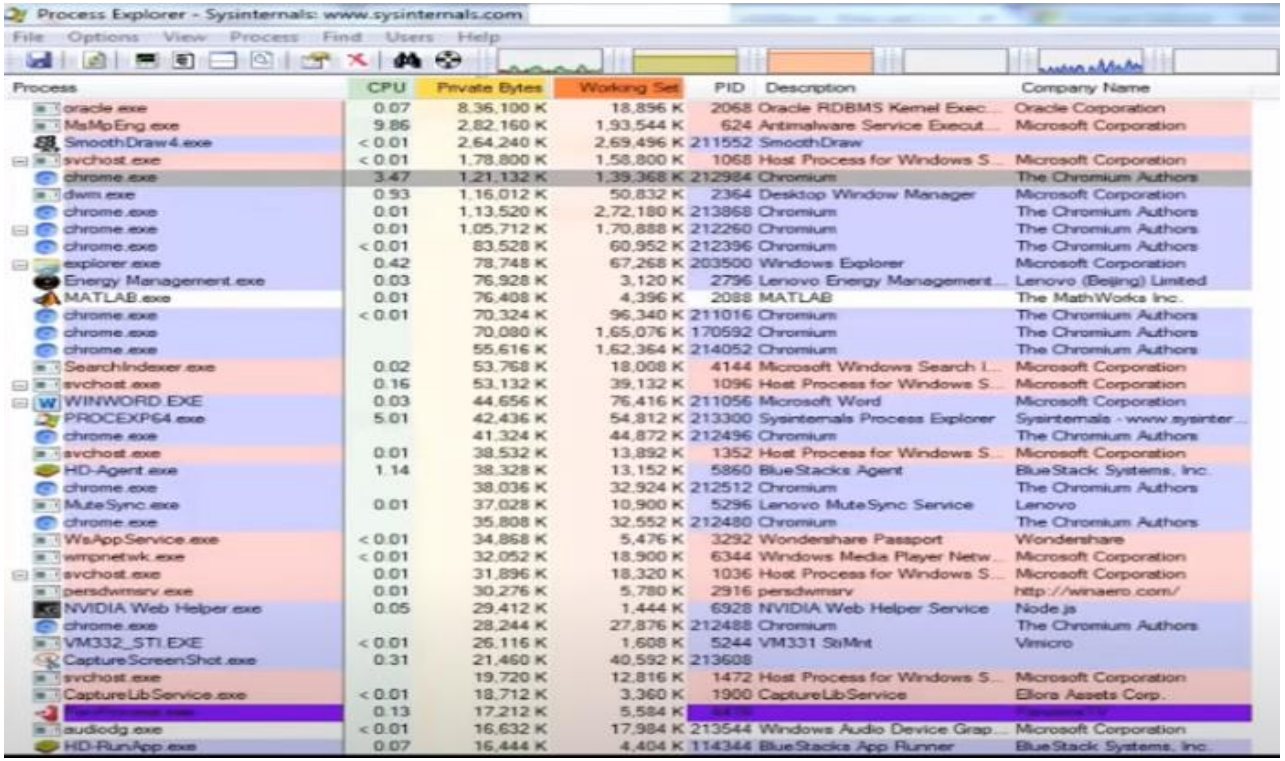
Applications



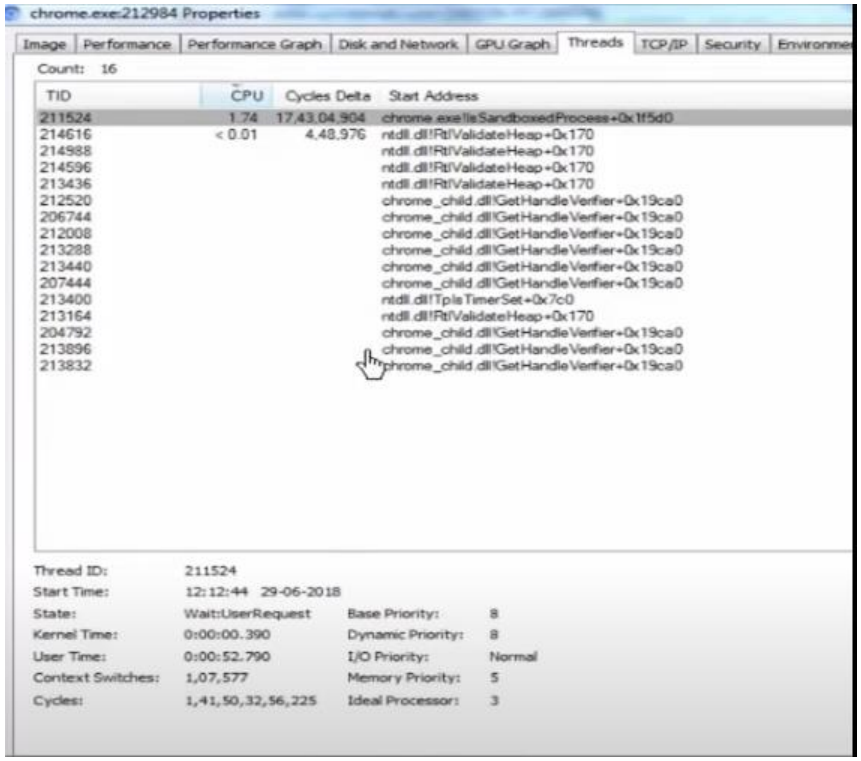
Processes


Applications
Processes
Services
Performance
Networking
Users
Image Name
User Name
CPU
Memory (Private Working Set)
Description
chrome.exe 00 97,276 K Chromium
chrome.exe 00 89,380 K Chromium
WINWORD.EXE *32 00 38,704 K Microsoft Word
explorer.exe 00 37,720 K Windows Explorer
chrome.exe 00 37,656 K Chromium
dwm.exe 01 28,660 K Desktop Window Manager
chrome.exe 00 19,976 K Chromium
chrome.exe 00 16,396 K Chromium
chrome.exe 00 16,228 K Chromium
CaptureScreenSh... 00 13,348 K CaptureScreenShot.exe
chrome.exe 00 10,804 K Chromium
HD-Agent.exe *32 00 6,408 K BlueStacks Agent
MuteSync.exe 00 4,760 K Lenovo MuteSync Service
taskmgr.exe 00 3,924 K Windows Task Manager
Show processes from all users
End Process

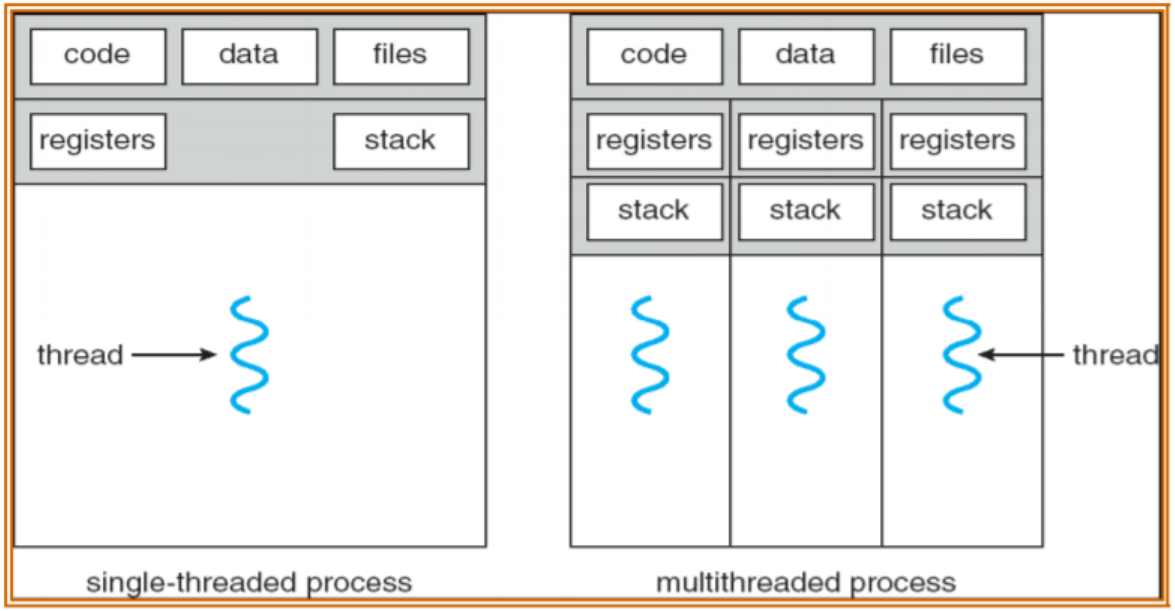
Processes


Process Explorer - Sysinternals: www.sysinternals.com
File Options View Process Find Users Help
Process CPU Private Bytes Working Set PID Description Company Name
oracle.exe 0.07 8,36,100 K 18,896 K 2068 Oracle RDBMS Kernel Exec... Oracle Corporation
MsMpEng.exe 9.86 2,82,160 K 1,93,544 K 624 Antimalware Service Execut... Microsoft Corporation
SmoothDraw4.exe < 0.01 2,64,240 K 2,69,496 K 211552 SmoothDraw
svchost.exe < 0.01 1,78,800 K 1,58,800 K 1068 Host Process for Windows S... Microsoft Corporation
chrome.exe 3.47 1,21,132 K 1,39,368 K 212984 Chromium The Chromium Authors
dwm.exe 0.93 1,16,012 K 50,832 K 2364 Desktop Window Manager Microsoft Corporation
chrome.exe 0.01 1,13,520 K 2,72,180 K 213868 Chromium The Chromium Authors
chrome.exe 0.01 1,05,712 K 1,70,888 K 212260 Chromium The Chromium Authors
chrome.exe < 0.01 83,528 K 60,952 K 212396 Chromium The Chromium Authors
explorer.exe 0.42 78,748 K 67,268 K 203500 Windows Explorer Microsoft Corporation
Energy Management.exe 0.03 76,928 K 3,120 K 2796 Lenovo Energy Management... Lenovo (Beijing) Limited
MATLAB.exe 0.01 76,408 K 4,396 K 2088 MATLAB The MathWorks Inc.
chrome.exe < 0.01 70,324 K 96,340 K 211016 Chromium The Chromium Authors
chrome.exe 70,080 K 1,65,076 K 170592 Chromium The Chromium Authors
chrome.exe 55,616 K 1,62,364 K 214052 Chromium The Chromium Authors
SearchIndexer.exe 0.02 53,768 K 18,008 K 4144 Microsoft Windows Search I... Microsoft Corporation
svchost.exe 0.16 53,132 K 39,132 K 1096 Host Process for Windows S... Microsoft Corporation
WINWORD.EXE 0.03 44,656 K 76,416 K 211056 Microsoft Word Microsoft Corporation
PROCEXP64.exe 5.01 42,436 K 54,812 K 213300 Sysinternals Process Explorer Sysinternals - www.sysinter...
chrome.exe 41,324 K 44,872 K 212496 Chromium The Chromium Authors
svchost.exe 0.01 38,532 K 13,892 K 1352 Host Process for Windows S... Microsoft Corporation
HD-Agent.exe 1.14 38,328 K 13,152 K 5860 BlueStacks Agent BlueStack Systems, Inc.
chrome.exe 38,036 K 32,924 K 212512 Chromium The Chromium Authors
MuteSync.exe 0.01 37,028 K 10,900 K 5296 Lenovo MuteSync Service Lenovo
chrome.exe 35,808 K 32,552 K 212480 Chromium The Chromium Authors
WsAppService.exe < 0.01 34,868 K 5,476 K 3292 Wondershare Passport Wondershare
wmpnetwk.exe < 0.01 32,052 K 18,900 K 6344 Windows Media Player Netw... Microsoft Corporation
svchost.exe 0.01 31,896 K 18,320 K 1036 Host Process for Windows S... Microsoft Corporation
persdwmsrv.exe 0.01 30,276 K 5,780 K 2916 persdwmsrv http://winaero.com/
NVIDIA Web Helper.exe 0.05 29,412 K 1,444 K 6928 NVIDIA Web Helper Service Node.js
chrome.exe 28,244 K 27,876 K 212488 Chromium The Chromium Authors
VM332_STI.EXE < 0.01 26,116 K 1,608 K 5244 VM331 StiMnt Vimicro
CaptureScreenShot.exe 0.31 21,460 K 40,592 K 213608
svchost.exe 19,720 K 12,816 K 1472 Host Process for Windows S... Microsoft Corporation
CaptureLibService.exe < 0.01 18,712 K 3,360 K 1900 CaptureLibService Ellora Assets Corp.
0.13 17,212 K 5,584 K
audiodg.exe < 0.01 16,632 K 17,984 K 213544 Windows Audio Device Grap... Microsoft Corporation
HD-RunApp.exe 0.07 16,444 K 4,404 K 114344 BlueStacks App Runner BlueStack Systems, Inc.

Thread


chrome.exe:212984 Properties
Image
Performance
Performance Graph
Disk and Network
GPU Graph
Threads
TCP/IP
Security
Count: 16
TID
CPU
Cycles Delta
Start Address
211524 1.74 17,43,04,904 chrome.exe!IsSandboxedProcess+0x1f5d0
214616 < 0.01 4,48,976 ntdll.dll!RtlValidateHeap+0x170
214988 ntdll.dll!RtlValidateHeap+0x170
214596 ntdll.dll!RtlValidateHeap+0x170
213436 ntdll.dll!RtlValidateHeap+0x170
212520 chrome_child.dll!GetHandleVerifier+0x19ca0
206744 chrome_child.dll!GetHandleVerifier+0x19ca0
212008 chrome_child.dll!GetHandleVerifier+0x19ca0
213288 chrome_child.dll!GetHandleVerifier+0x19ca0
213440 chrome_child.dll!GetHandleVerifier+0x19ca0
207444 chrome_child.dll!GetHandleVerifier+0x19ca0
213400 ntdll.dll!TpIsTimerSet+0x7c0
213164 ntdll.dll!RtlValidateHeap+0x170
204792 chrome_child.dll!GetHandleVerifier+0x19ca0
213896 chrome_child.dll!GetHandleVerifier+0x19ca0
213832 chrome_child.dll!GetHandleVerifier+0x19ca0
Thread ID: 211524
Start Time: 12:12:44 29-06-2018
State: Wait:UserRequest
Base Priority: 8
Kernel Time: 0:00:00.390
Dynamic Priority: 8
User Time: 0:00:52.790
I/O Priority: Normal
Context Switches: 1,07,577
Memory Priority: 5
Cycles: 1,41,50,32,56,225
Ideal Processor: 3

Thread



Single-threaded and multithreaded processes

Benefits

The benefits of multi threaded programming can be broken down into four major categories:

Responsiveness

Multi threading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user. A multi threaded web browser could still allow user interaction in one thread while an image was being loaded in another thread.

Resource sharing

By default, threads share the memory and the resources of the process to which they belong. The benefit of sharing code and data is that it allows an application to have several different threads of activity within the same address space.

Benefits

Economy

- Allocating memory and resources for **process creation is costly**. Because threads share resources of the process to which they belong, it is more economical to create and context-switch threads.
- It is much more time consuming to create and manage processes than threads. In Solaris, for example, creating a process is about thirty times slower than is creating a thread, and context switching is about five times slower.

Utilization of multiprocessor architectures

- The benefits of multithreading can be greatly increased in a multiprocessor architecture, where threads may be running in parallel on different processors.
- A single threaded process can only run on one CPU, no matter how many are available.
- Multithreading on a multi-CPU machine increases concurrency.

User Level Thread

User threads are supported above the kernel and are managed without kernel support i.e., they are implemented by **thread library** at the user level. These are the threads that application programmers use in their programs.

- The library provides support for **thread creation, scheduling and management** with no support from the kernel.
- Because the kernel is unaware of user-level threads, all thread creation and scheduling are done in user space without the need for kernel intervention.
- Therefore, **user-level threads are generally fast to create and manage**. User-thread libraries include POSIX Pthreads, Mach C-threads, and Solaris 2 UI-threads.

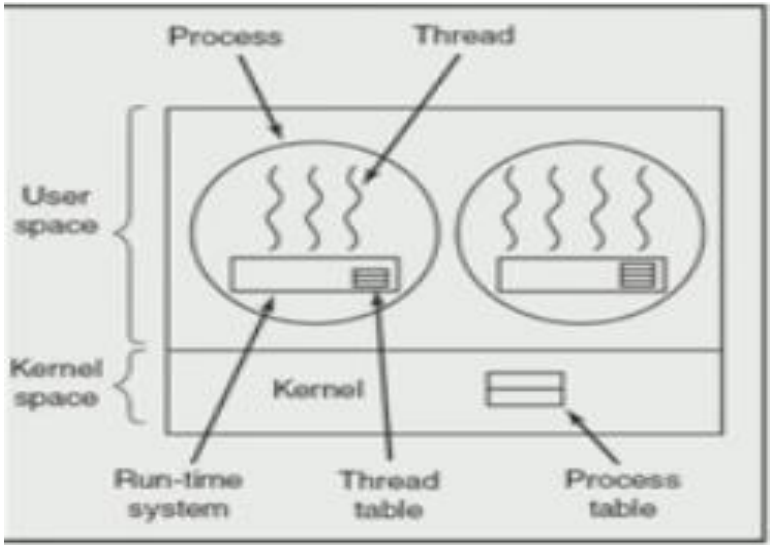
User Level Thread

Advantage:

- Fast
- Thread Management happens with thread library.

Drawback:

- Lack of Coordination between Kernel and Thread
- If one thread invokes system call, all threads has to wait



Kernel Level Thread

Kernel threads are supported and managed directly by the operating system.

- The kernel performs **thread creation, scheduling** and management in kernel space.
- Since thread management is done by the operating system, kernel threads are generally slower to create and manage than are user threads.
- Also, in a multiprocessor environment, the kernel can schedule threads on different processors. Most contemporary operating systems—including Windows NT, Windows 2000, Solaris 2,BeOS, and Tru64 UNIX (formerly Digital UNIX)—support kernel threads

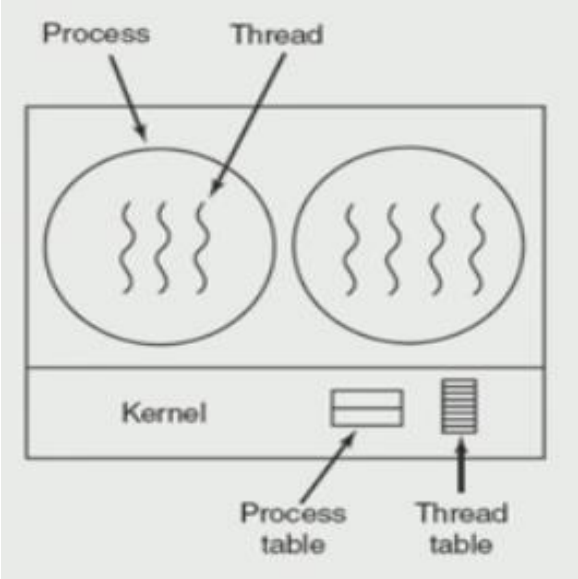
Kernel Level Thread

Advantage:

- Scheduler can decide to give more time to process having more threads
- IF a Thread invokes a system call the remaining thread will run.

Drawback:

- Slow
- Overhead to manage thread and process



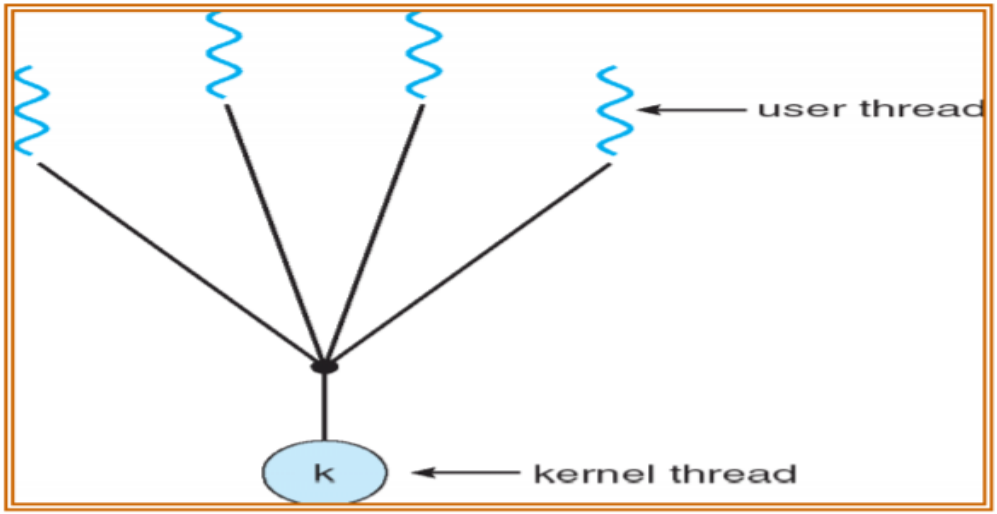
Multithreading Models

Some operating system provide a combined user level thread and Kernel level thread facility. Eg. Solaris.

Three common ways of establishing this relationship are:

- **Many-to-One Model**
- **One-to-one Model**
- **Many-to-Many Model**

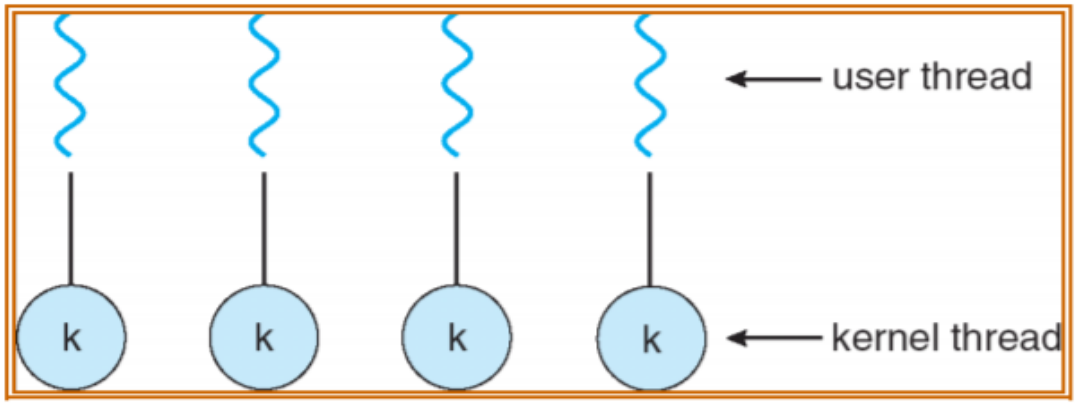
Many-to-One Model



Many-to-One Model

- The many-to-one model maps **many user-level threads to one kernel thread**.
- Thread management is done by the **thread library in user space**, so it is efficient; but the entire process will block if a thread makes a blocking system call.
- Also, because only one thread can access the kernel at a time, multiple threads are unable to run in parallel on multiprocessors. **Green threads**—a thread library available for Solaris 2—uses this model.

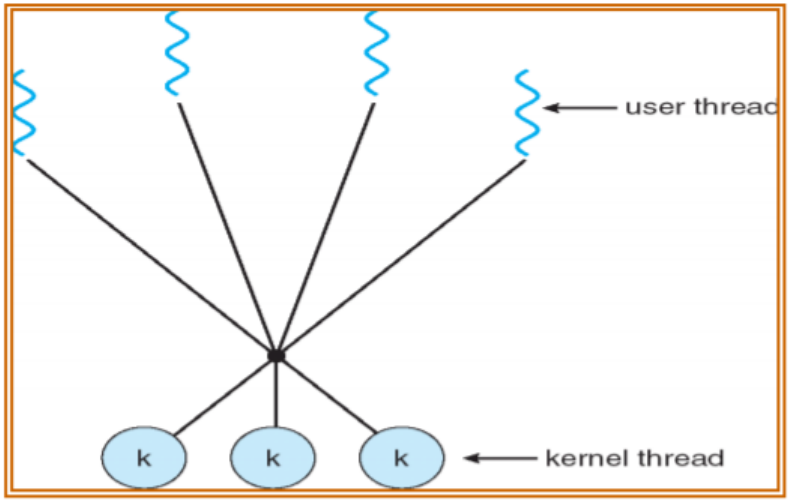
One-to-One Model



One-to-One Model

- The one-to-one model maps **each user thread to a kernel thread**. It provides more **concurrency than the many-to-one model** by allowing another thread to run when a thread makes a blocking system call.
- It also allows multiple threads to run in **parallel** on multiprocessors.
- The only drawback to this model is that creating a user thread requires creating the corresponding kernel thread.
- **Linux**, along with the family of Windows operating systems—including **Windows 95, 98, NT, 2000, and OS/2**— implement the one-to-one model.

Many-to-Many Model



Many-to-Many Model

- The many-to-many model multiplexes **many user-level threads to a smaller or equal number of kernel threads**.
- The one-to-one model allows for greater concurrency, but the developer has to be careful not to create too many threads within an application.
- The many-to-many model suffers from neither of these shortcomings: Developers can create as many user threads as necessary, and the corresponding kernel threads can run in parallel on a multiprocessor. Also, when a thread performs a blocking system call, the kernel can schedule another thread for execution. **Solaris 2, IRIX, HP-UX and Tru64 UNIX support this model.**

Identification of Core:(Windows Management Interface Command)

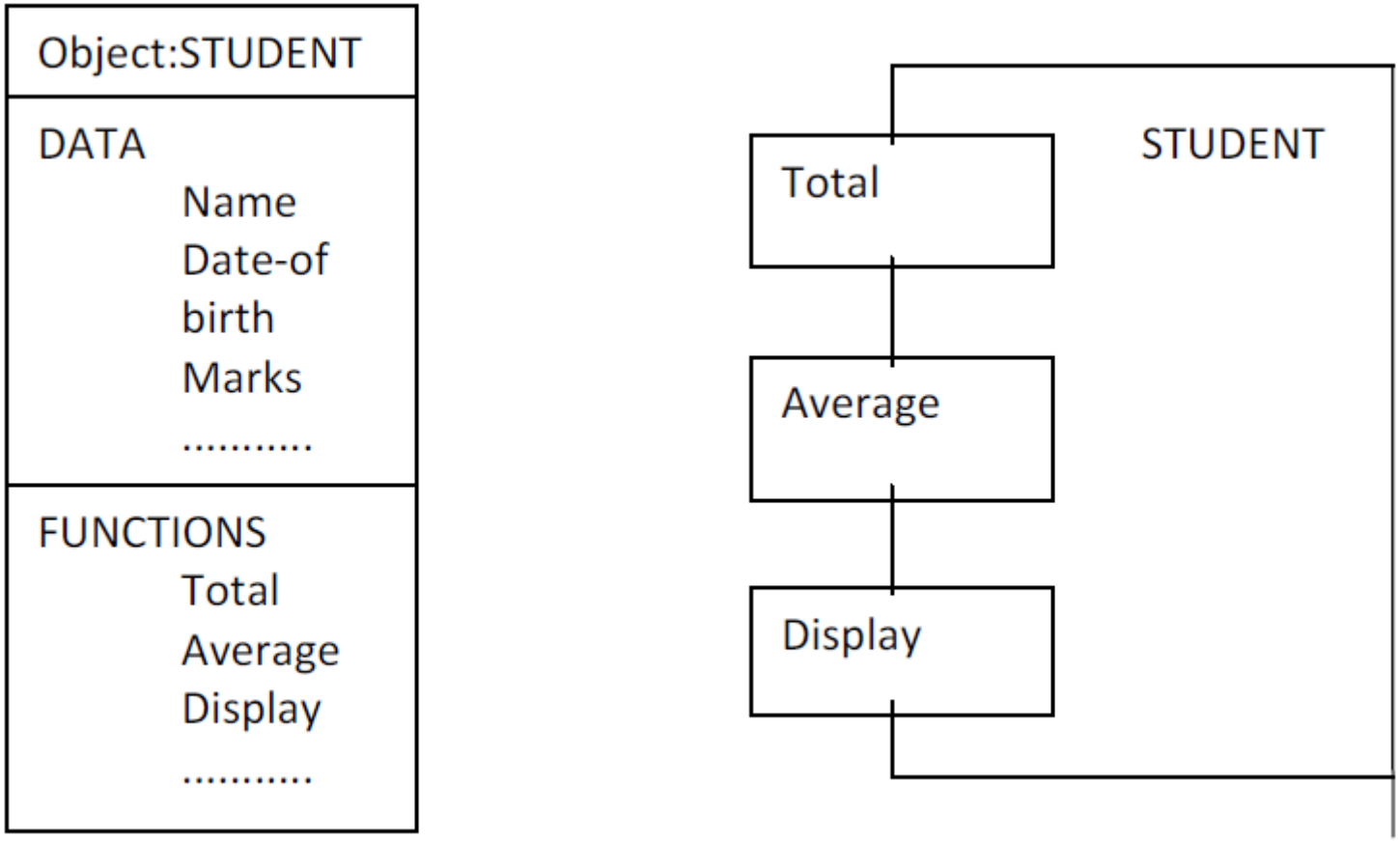
```
C:\Users\Ajitha>wmic
wmic:root\cli>CPU Get NumberOfCores
NumberOfCores
4

wmic:root\cli>CPU Get NumberOfCores,NumberOFLogicalProcessors
NumberOfCores  NumberOfLogicalProcessors
4              8

wmic:root\cli>
```

Difference between Process and Thread

Sl.No	Process	Thread
1	Process means any program is in execution.	Thread means segment of a process.
2	It takes more time for creation.	It takes less time for creation.
3	Process is called heavy weight process.	Thread is called light weight process.
4	Process switching needs interaction with operating system	Thread switching does not need to interact with operating system
5	In multiple processing environments each process executes the same code but has its own memory and file resources	All threads can share same set of open files, Child processes

Object

- **Objects are the basic run time entities in an object-oriented system.**
- **They may** represent a person, a place, a bank account,a table of data or any item that the program has to handle.
- Programming problem is analysed in terms of objects and the nature of communication between them.
- When a program is executed,the objects interact by sending messages to one another.
- For example, if 'customer' and 'account' are the two objects in a program, then the customer object may send a message to the account object.
- Each object contains data and code to manipulate the data.



Two ways of representing an object

- Real-world objects share two characteristics: They all have *state and behavior.*
- Desktop lamp may have only two possible states (on and off) and two possible behaviors (turn on, turn off), but your desktop radio might have additional states (on, off, current volume, current station) and behavior (turn on, turn off, increase volume, decrease volume, seek, scan, and tune).
- You may also notice that some objects, in turn, will also contain other objects.
- These real-world observations all translate into the world of object-oriented programming.

Device Management

- Device management of operating system manages hardware devices via their respective drivers
- It deals with the management of the I/O devices such as a keyboard, magnetic tape, disk, printer, microphone, USB ports, scanner, camcorder etc.as well as the supporting units like control channels.
- The basics of I/O devices can fall into 3 categories:
  - **Block device:** it stores information in fixed-size block. example, disks.
  - **Character device:** it delivers or accepts a stream of characters. example printers, keyboards
  - **Network device:** For transmitting data packets.

Activities of Device Management

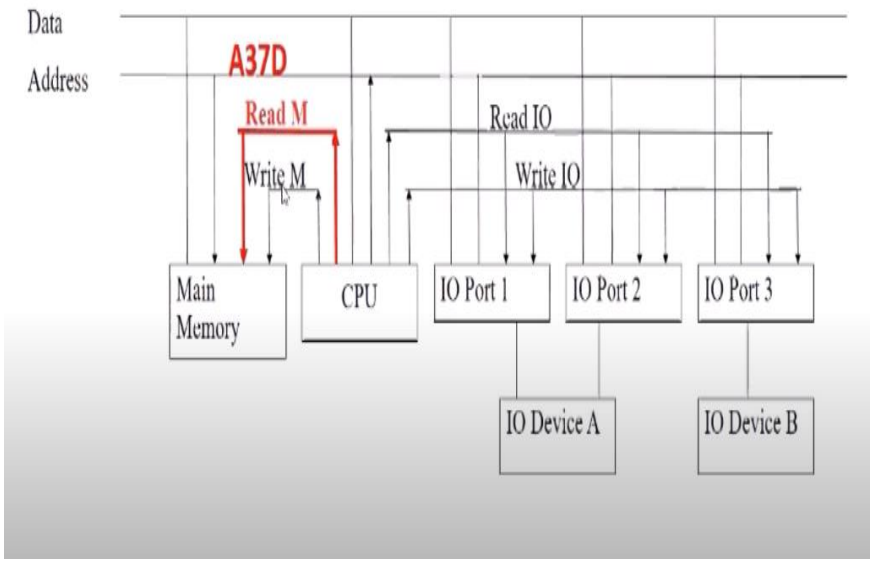
- It keeps track of all the devices. The program responsible for keeping track of all the devices is known as I/O controller.
- It provides a uniform interface to access devices with different physical characteristics.
- It allocates the devices in an efficient manner.
- It deallocates the devices after usage.
- It decides which process gets the device and how much time it should be used.
- It optimizes the performance of each individual device.

Approaches

- Direct I/O
- Memory Mapped I/O
- Direct Memory Access

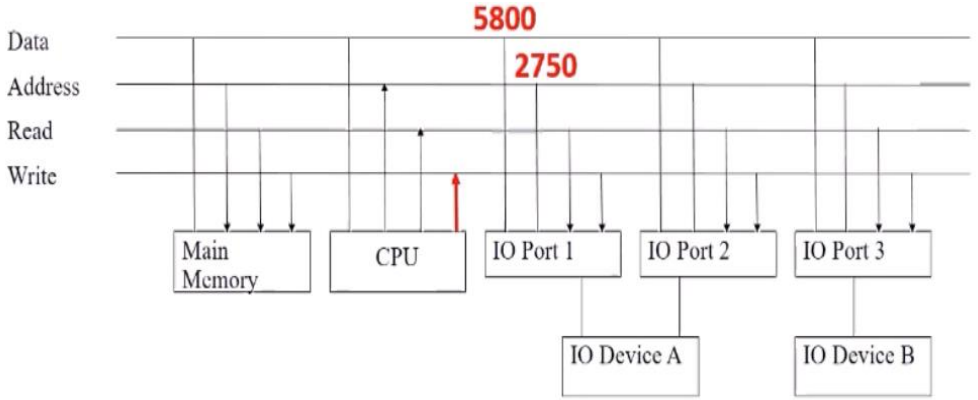
Direct I/O

- CPU software explicitly transfer data to and from the controller's data registers.
- Separate Control signal for Memory and I/O Devices
- Common Address and Data Bus for I/O Devices and Memory



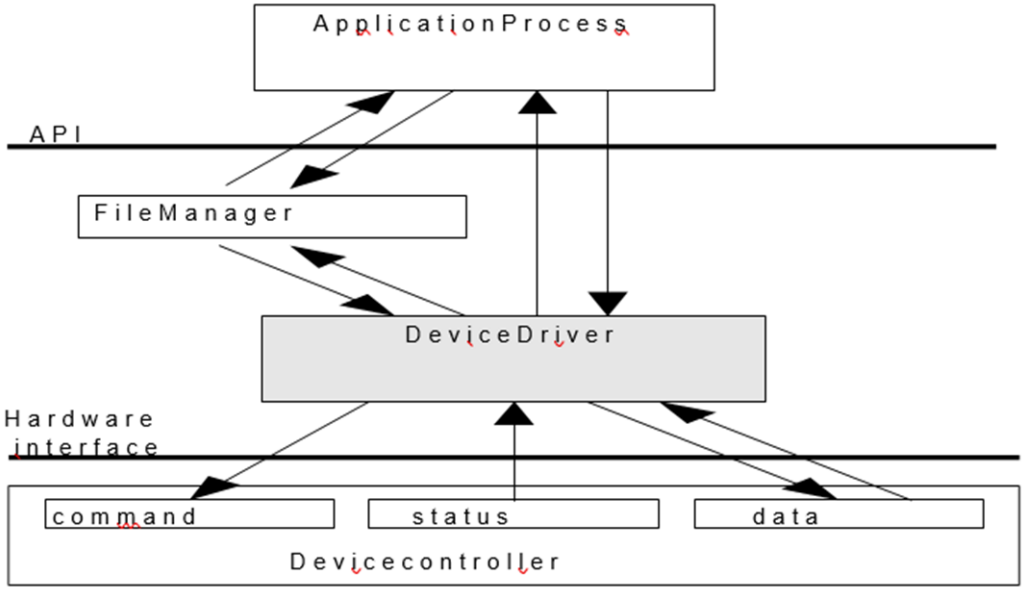
MEMORY MAPPED I/O

- Device addressing simplifies the interface (device seen as a range of memory locations)
- Common Control signal for Memory and I/O Devices
- Common Address and Data Bus for I/O Devices and Memory



Consider the address space for Main Memory is 1000 – 2500
And address space for I/O devices is 2501 - 3500

I/O System Organization



I/O System Organization

- An application process uses a device by issuing commands and exchanging data with the device management (device driver).
- Device driver responsibilities:
  - Implement communication APIs that abstract the functionality of the device
  - Provide device dependent operations to implement functions defined by the API
- API should be similar across different device drivers, reducing the amount of info an application programmer needs to know to use the device

I/O System Organization

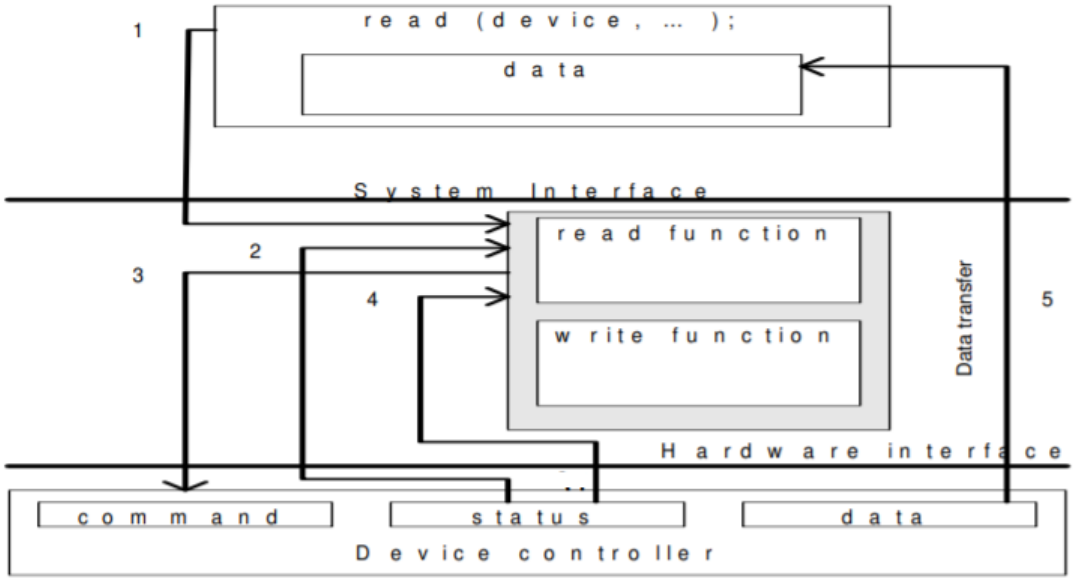
Since each device controller is specific to a particular device, the device driver implementation will be device specific, to

- Provide correct commands to the controller
- Interpret the controller status register (CSR) correctly
- Transfer data to and from device controller data registers as required for correct device operation

I/O with polling

- Each I/O operation requires that the software and hardware coordinate their operations to accomplish desired effect
- In direct I/O polling this coordination is done in the device driver;
- While managing the I/O, the device manager will poll the busy/done flags to detect the operation's completion; thus, the CPU starts the device, then polls the CSR to determine when the operation has completed
- With this approach is difficult to achieve high CPU utilization, since the CPU must constantly check the controller status

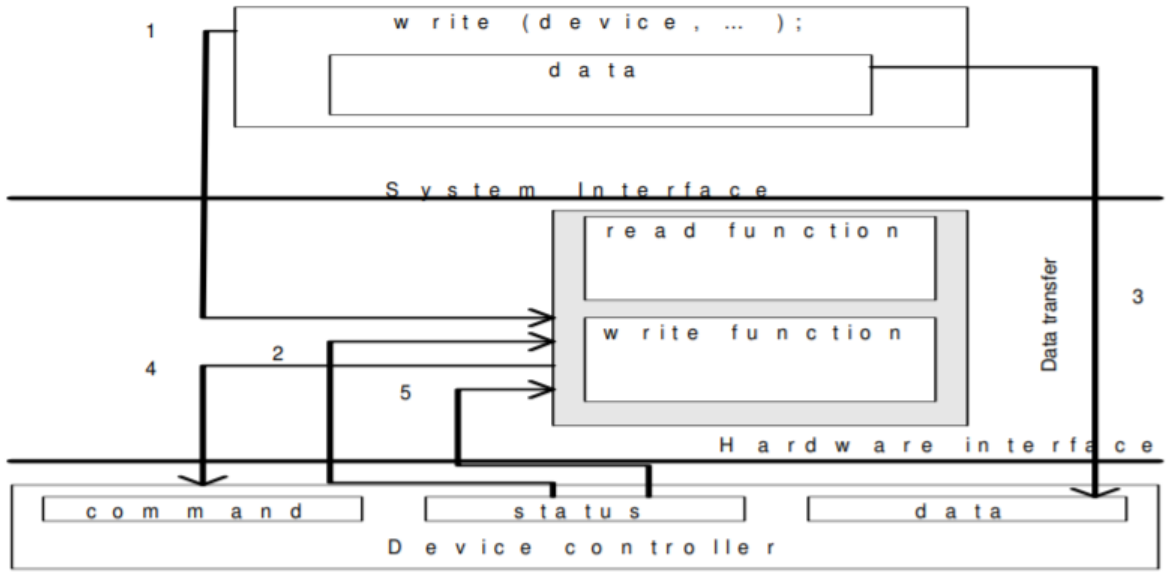
I/O with polling – read



I/O with polling-read

- Application process requests a read operation
- The device driver queries the CSR to determine whether the device is idle; if device is busy, the driver waits for it to become idle
- The driver stores an input command into the controller's command register, thus starting the device
- The driver repeatedly reads the content of CSR to detect the completion of the read operation
- The driver copies the content of the controller's data register(s) into the main memory user's process's space.

I/O with polling – write

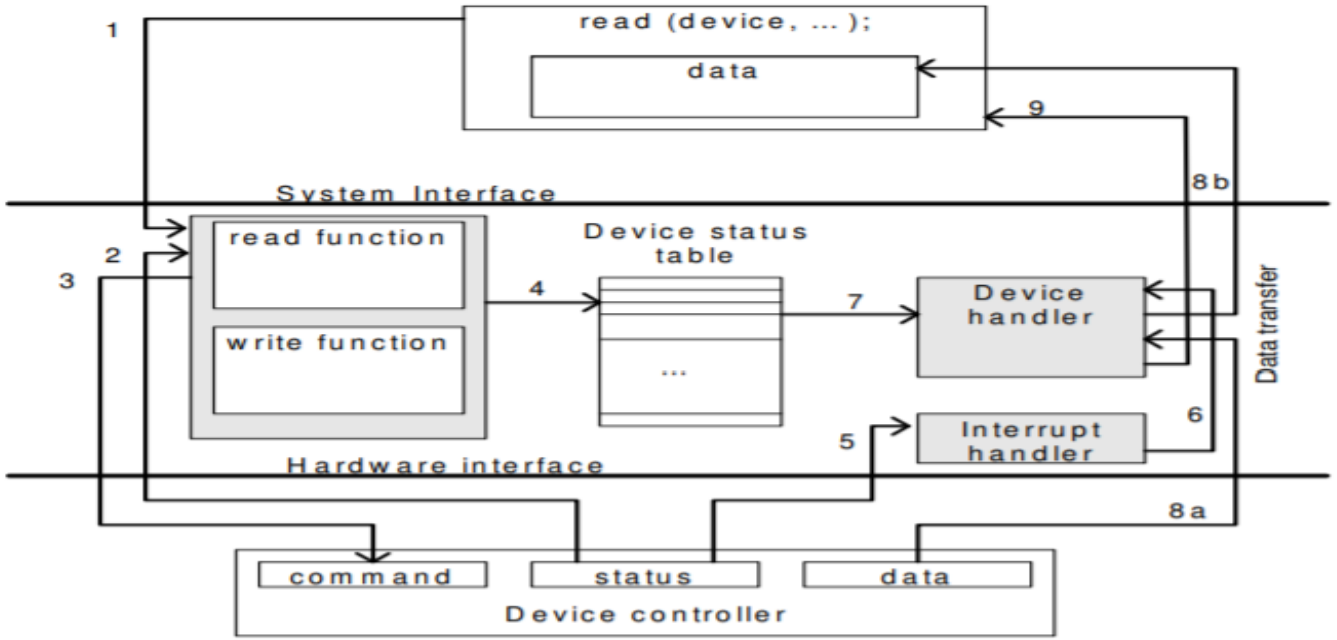


I/O with polling-write

- The application process requests a write operation
- The device driver queries the CSR to determine if the device is idle; if busy, it will wait to become idle
- The device driver copies data from user space memory to the controller's data register(s)
- The driver stores an output command into the command register, thus starting the device
- The driver repeatedly reads the CSR to determine when the device completed its operation.

Interrupt driven I/O

- In a multiprogramming system the wasted CPU time (in polled I/O) could be used by another process; because the CPU is used by other processes in addition to the one waiting for the I/O operation completion.
- This may be remedied by use of interrupts
- The reason for incorporating the interrupts into a computer hardware is to eliminate the need for a device driver to constantly poll the CSR
- Instead polling, the device controller "automatically" notifies the device driver when the operation has completed.


1
read (device, ...);
data
9
8b
System Interface
2
3
read function
write function
Device status table
4
...
7
Device handler
Data transfer
6
5
Interrupt handler
Hardware interface
8a
command
status
data
Device controller

Interrupt driven I/O

- The application process requests a read operation
- **T**he device driver queries the CSR to find out if the device is idle; if busy, then it waits until the device becomes idle
- **T**he driver stores an input command into the controller's command register, thus starting the device
- When this part of the device driver completes its work, it saves information regarding the operation it began in the device status table; this table contains an entry for each device in system; the information written into this table contains the return address of the original call and any special parameters for the I/O operation; the CPU, after doing this, can be used by other program, so the device manager invokes the scheduler part of the process manager. It then terminates

Interrupt driven I/O

- The device completes the operation and interrupts the CPU, therefore causing an *interrupt handler* to run
- The interrupt handler determines which device caused the interrupt; it then branches to the *device handler* for that device
- The device driver retrieves the pending I/O status information from the device status table
- **(a,b) T**he device driver copies the content of the controller's data register(s) into the user process's space
- The device handler returns the control to the application process (knowing the return address from the device status table). Same sequence (or similar) of operations will be accomplished for an output operation.

Buffering

Buffering is a technique by which a device manager keeps the slower I/O devices busy when a process is not requiring I/O operations.

A buffer is a memory area that stores data being transferred between two devices or between a device and an application.

- **Input buffering** is the process of reading the data into the primary memory before the process requests it.
- **Output buffering** is the process of saving the data in the memory and then writing it to the device while the process continues its execution.

Hardware level buffering

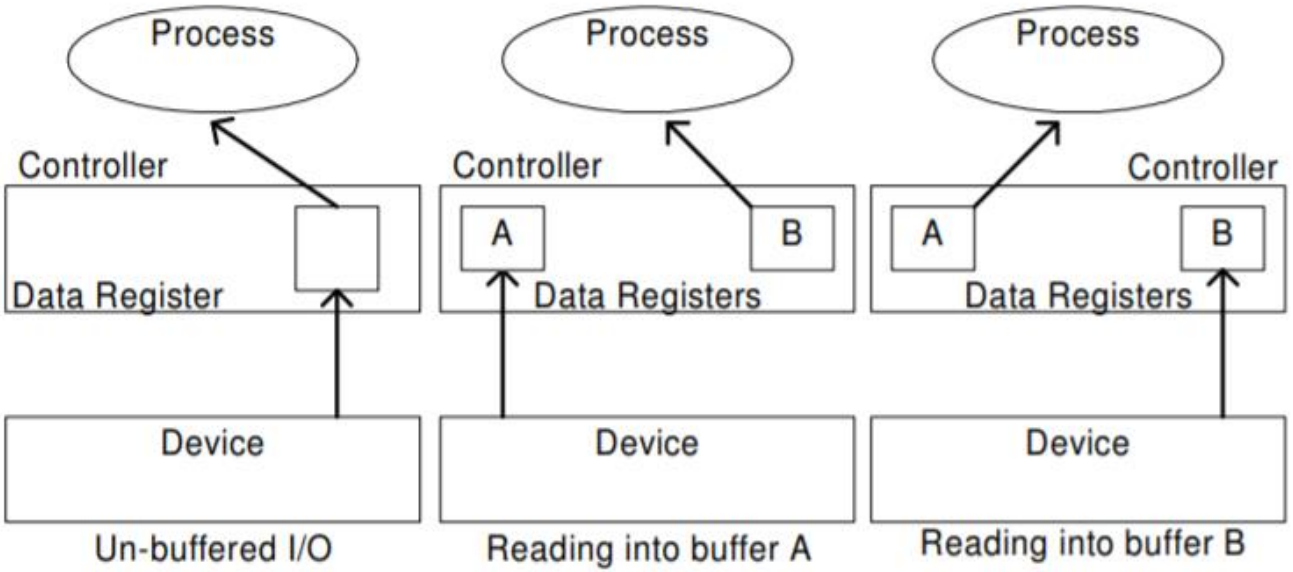
Consider a simple character device controller that reads a single byte form a modem for each input operation.

- Normal operation: read occurs, the driver passes a read command to the controller; the controller instructs the device to put the next character into one-byte data controller's register; the process calling for byte waits for the operation to complete and then retrieves the character from the data register

Add a hardware buffer to the controller to decrease the amount of time the process has to wait

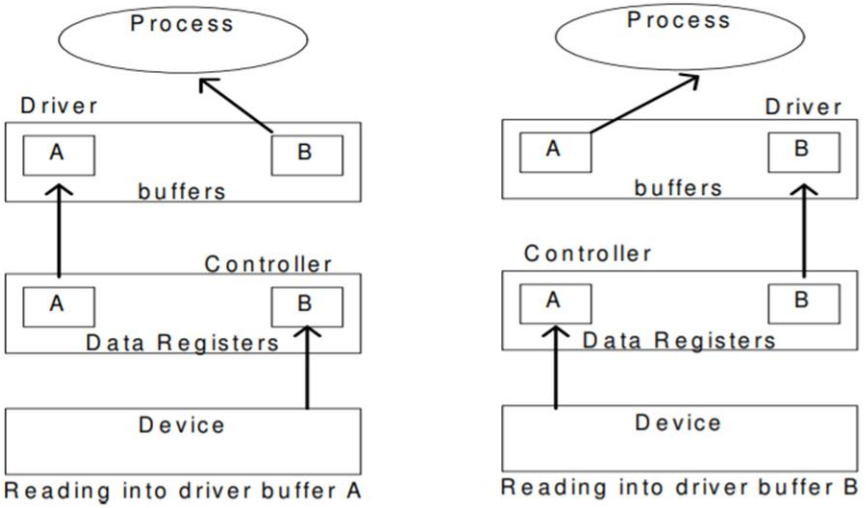
- Buffered operation: the next character to be read by the process has already been placed into the data register, even the process has not yet called for the read operation

Hardware level buffering



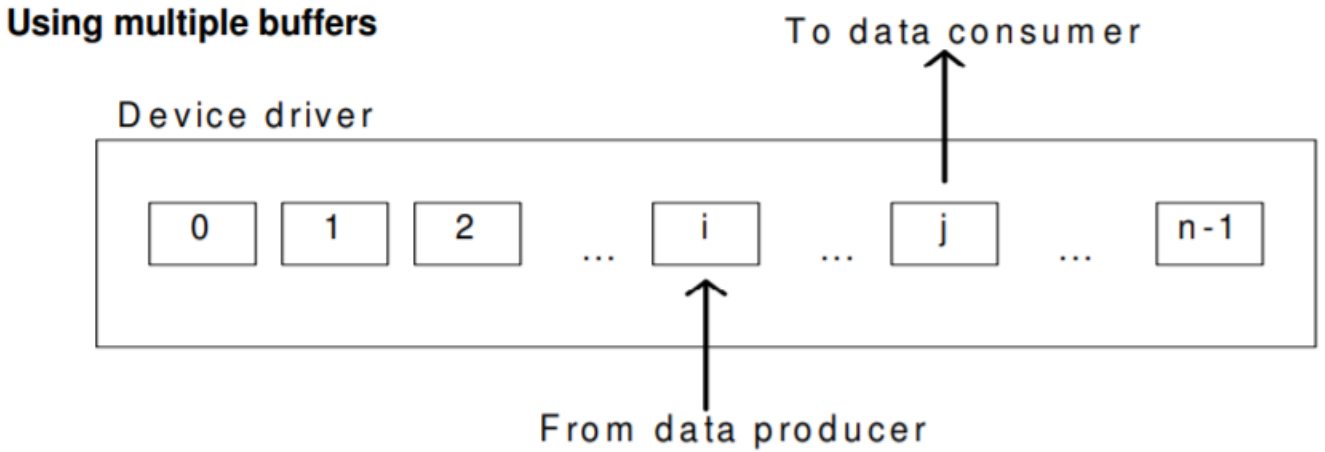
Driver level buffering

This is generally called double buffering. One buffer is for the driver to store the data while waiting for the higher layers to read it. The other buffer is to store data from the lower level module. This technique can be used for the block-oriented devices (buffers must be large enough to accommodate a block of data).



Driver level buffering

- The number of buffers is extended from two to n. The data producer is writing into buffer i while the data consumer is reading from buffer j.
- In this configuration buffers j+1 to n-1 and 0 to i-1 are full. This is known as circular buffering technique.



Device driver

- It is a software program that controls a particular type of device attached to the computer. It provides an interface to the hardware devices without the requirement to know the precise information about the hardware.
- A device driver communicates with the device through a bus or communication sub system.

Responsibilities

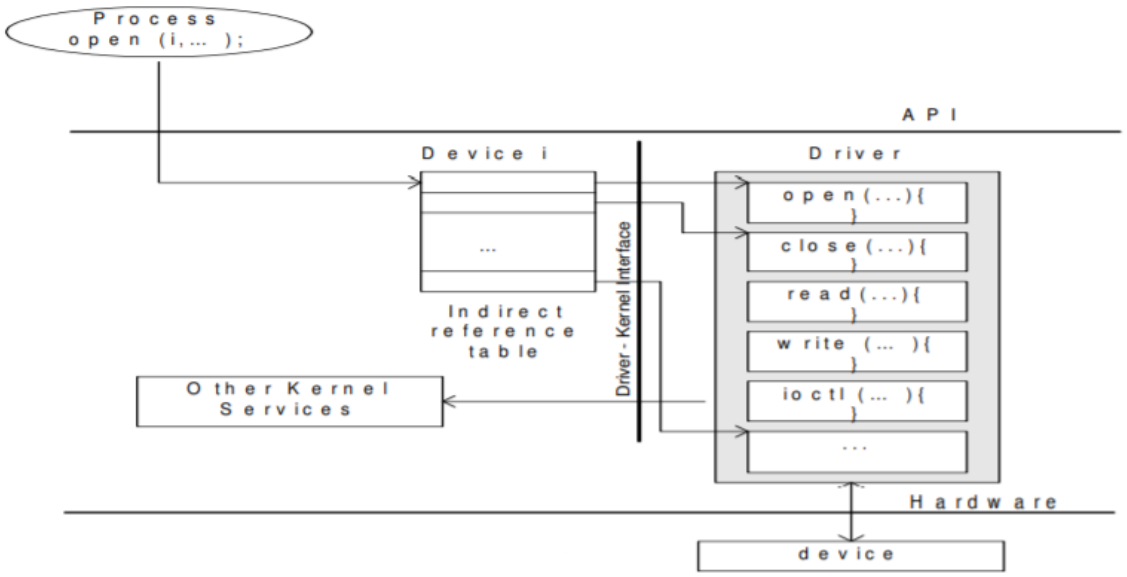
- Initialize devices
- Interpreting the commands from the operating system
- Manage data transfers
- Accept and process interrupts
- Maintain the integrity of driver and kernel data structures

Two ways of dealing with the device drivers

- Old way: Driver is part of the operating system, to add a new device driver, the whole OS must have been complied
- Modern way: Drivers installation is allowed without re-compilation of the OS by using reconfigurable device drivers; the OS dynamically binds the OS code to the driver functions.

  - A reconfigurable device driver has to have a fixed, standardized API
  - The Kernel has to provide an interface to the device driver to allocate/de-allocate space for buffers, manipulate tables in the kernel etc.

Reconfigurable device drivers

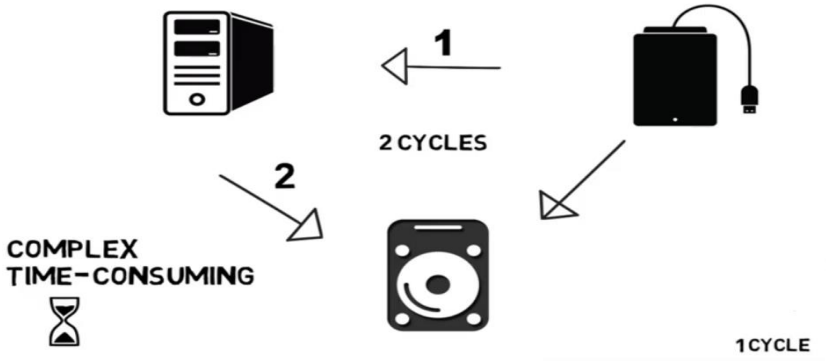


The driver - kernel interface

- The OS uses an indirect reference table to access the different driver entry points, based on the device identifier and function name. The indirect reference table is filled with appropriate values whenever the device driver loads
- When a process performs a system call, the kernel passes the call onto the device driver via the indirect reference table.

DIRECT MEMORY ACCESS

- Involves designing of hardware to avoid the CPU perform the transfer of information between the device.
- Normally Data transfer from I/O to Memory happens in two cycle.
  - I/O to CPU
  - CPU to Memory
- Using DMA data can be transferred directly from I/O to Memory



Direct Memory Access

- It consists of Four Blocks
  - CPU
  - DMAC(Direct Memory Access Controller)
  - Disk controller
  - Memory

Steps in Direct Memory Access

- CPU will set the registers of DMAC(ie) Cpu programs the DMAC by setting its Registers
- CPU instruct the disk controller to read the data from the drive and store it in buffer.
- DMAC request disk controller to transfer data to memory
- Disk Controller will transfer the data to memory
- Disk Controller will send the acknowledgement to the DMA Controller after transferring the data.
- After getting the acknowledgement the DMA Controller will decrement the count.
- Once the Byte count is zero, it tells to CPU the Data transfer is over.

DIRECT MEMORY ACCESS