ADV. OPERATING SYSTEMS

OPERATING SYSTEM OVERVIEW

What is an Operating System?

Operating systems are those ***programs*** *that* ***interface*** *the machine with the applications programs. The main function of these systems is to* ***dynamically allocate*** *the* ***shared*** *system* ***resources*** *to the executing programs. As such, research in this area is clearly concerned with the* ***management*** *and* ***scheduling*** *of* ***memory****,* ***processes****, and* ***other devices****.*

—WHAT CAN BE AUTOMATED?: THE COMPUTER SCIENCE AND ENGINEERING RESEARCH STUDY,
MIT Press, 1980

What is an Operating System?

❖ An Operating System is a program or collection of programs that makes it easier for us to use a computer.

❖ An Operating System provides simpler abstraction of the underlying hardware.

❖ An Operating System is resource manager.

Examples:

- DOS, OS/2, Windows XP, Windows 2000
- Ubuntu, FreeBSD, Fedora, Solaris, Mac OS
- iOS, Android, Symbian OS, Lynx OS

Objectives of an Operating System

A program that controls the execution of application programs

An interface between applications and hardware

Main objectives of an OS:

- Convenience
- Efficiency
- Ability to evolve

COMPUTER HARDWARE AND SOFTWARE INFRASTRUCTURE

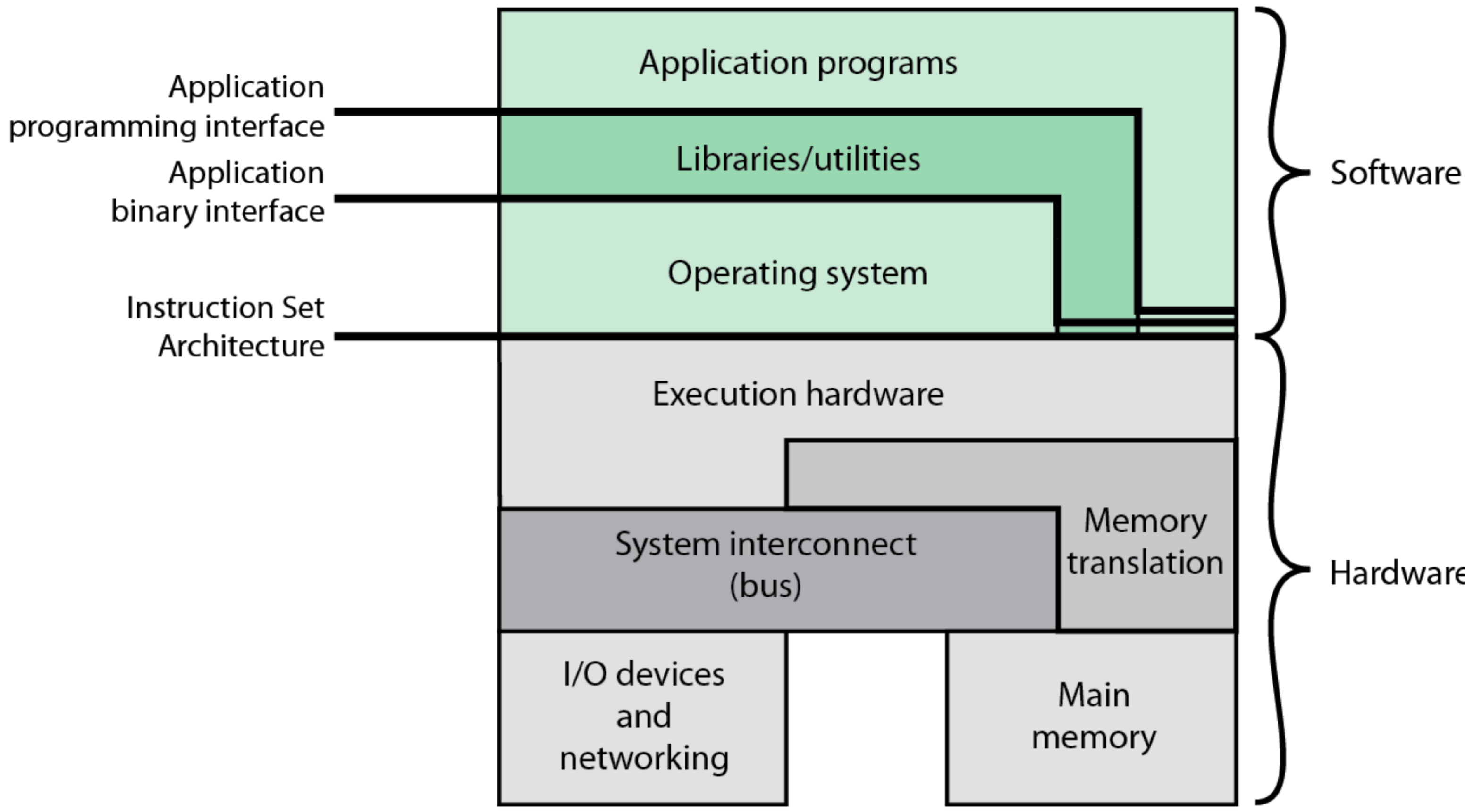

Application programming interface
Application binary interface
Instruction Set Architecture
Application programs
Libraries/utilities
Operating system
Execution hardware
System interconnect (bus)
Memory translation
I/O devices and networking
Main memory
Software
Hardware

Figure 2.1 Computer Hardware and Software Infrastructure

OPERATING SYSTEM SERVICES

- Program development
- Program execution
- Access I/O devices
- Controlled access to files
- System access
- Error detection and response
- Accounting

KEY INTERFACES

1. Instruction Set Architecture (**ISA**)
2. Application Binary Interface (**ABI**)
3. Application Programming Interface (**API**)

API VS ABI

- ❖An API is a contract between pieces of source code: It defines the parameters to a function, the function's return value, and attributes such as whether inheritance is allowed.
- ❖An API is enforced by the compiler: An API is instructions to the compiler about what source code can and cannot do. We also often speak about the API in terms of the prerequisites, behavior, and error conditions of functions. In that sense, an API is also consumed by humans: An API is instructions to a programmer about what functions expect and do.

API VS ABI

- ❖An ABI is a contract between pieces of binary code: It defines the mechanisms by which functions are invoked, how parameters are passed between caller and callee, how return values are provided to callers, how libraries are implemented, and how programs are loaded into memory.
- ❖An ABI is enforced by the linker: An ABI is the rules about how unrelated code must work together. An ABI is also rules about how processes coexist on the same system. For example, on a Unix system, an ABI might define how signals are executed, how a process invokes system calls, what endianness is used, and how stacks grow. In that sense, an ABI is a set of rules enforced by the operating system on a specific architecture.

https://www.quora.com/What-exactly-is-an-Application-Binary-Interface-ABI

http://stackoverflow.com/questions/2171177/what-is-application-binary-interface-abi

THE ROLE OF AN OS

- ❖A computer is a set of resources for the movement, storage, and processing of data and for the control of these functions.
- ❖The OS is responsible for managing these resources
- ❖*Normally, we think of a control mechanism as something external to that which is controlled.*

  Example: Heating System and Thermostat

OPERATING SYSTEM AS SOFTWARE

❖ Functions in the same way as ordinary computer software i.e. Program, or suite of programs, executed by the processor

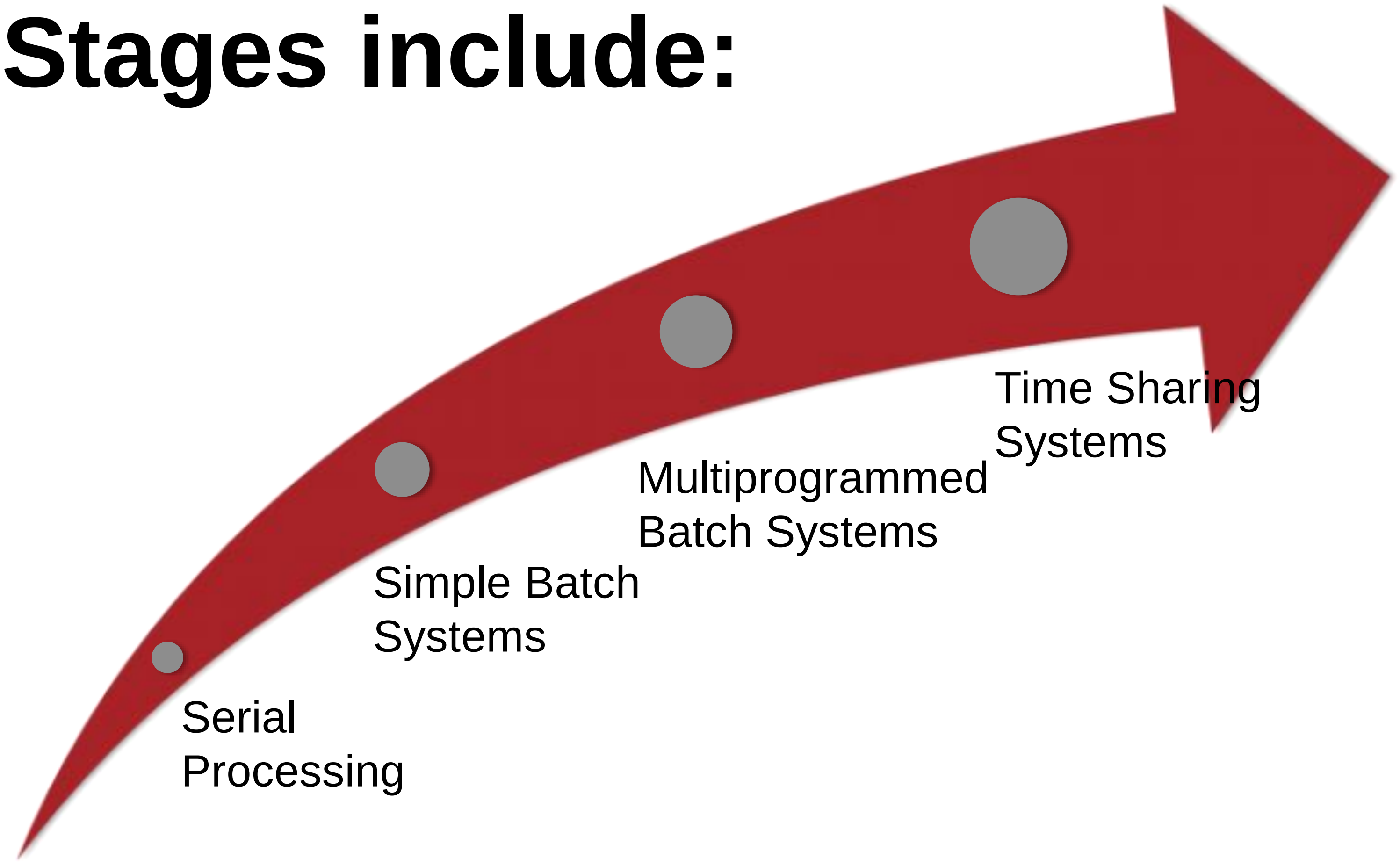
EVOLUTION OF OPERATING SYSTEMS

A major OS will evolve over time for a number of reasons:

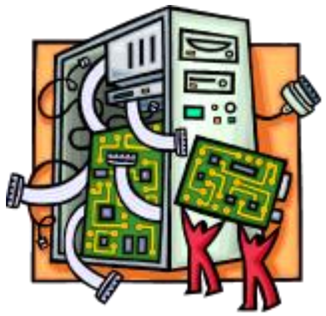
- Hardware Upgrades
- New Types of Hardware
- New Services
- Bug Fixes

EVOLUTION OF OPERATING SYSTEMS

- Stages include:



SERIAL PROCESSING



EARLIEST COMPUTERS:

- No operating system
  - Programmers interacted directly with the computer hardware
- Computers ran from a console with display lights, toggle switches, some form of input device, and a printer
- Users have access to the computer in ―series‖

PROBLEMS:

Scheduling

- Most installations used a hardcopy sign-up sheet to reserve computer time
- Time allocations could run short or long, resulting in wasted computer time

Setup time

- A considerable amount of time was spent just on setting up the program to run

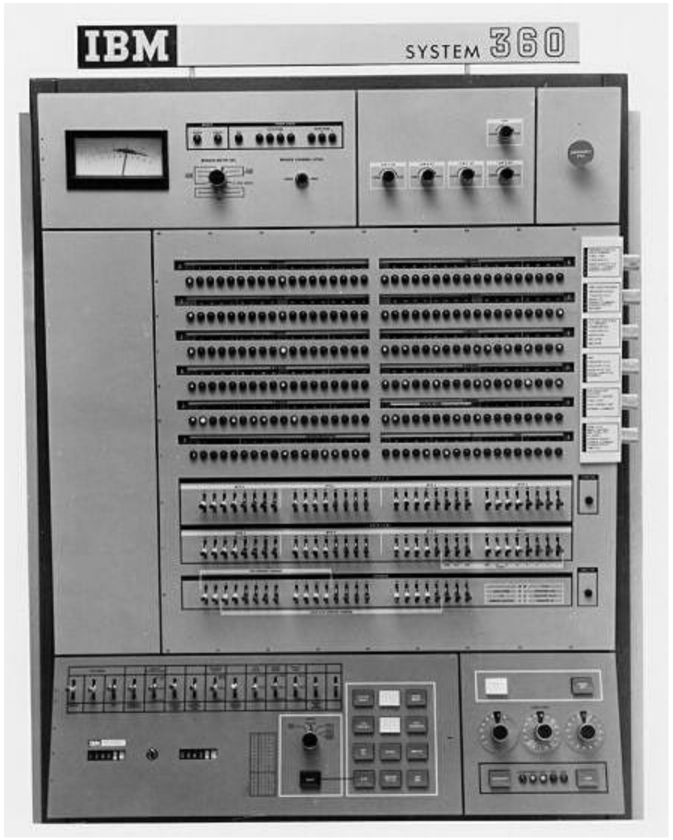
Slide 15

Introduction

NO SMOKING

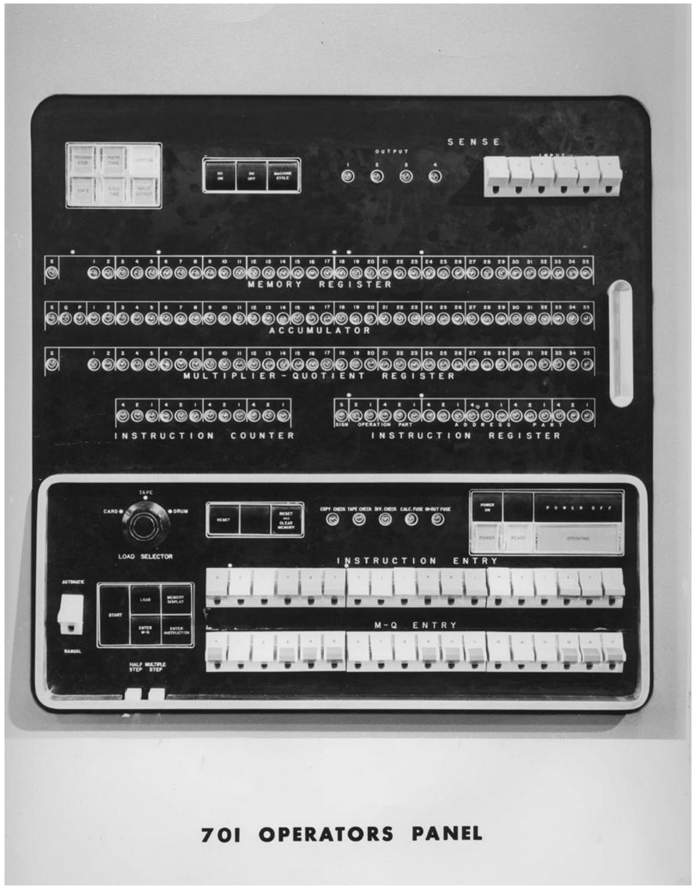
Slide 17

Lecture Notes: Introduction



IBM 7094 (Early 1960's)

IBM 701 Console



SIMPLE BATCH SYSTEMS

Early computers were very expensive

- Important to maximize processor utilization

Monitor

- User no longer has direct access to processor
- Job is submitted to computer operator who batches them together and places them on an input device
- Program branches back to the monitor when finished

MONITOR POINT OF VIEW

- ❖Monitor controls the sequence of events
- ❖*Resident Monitor* is software that always resides in memory
- ❖Monitor reads in job and gives control
- ❖Job returns control to monitor

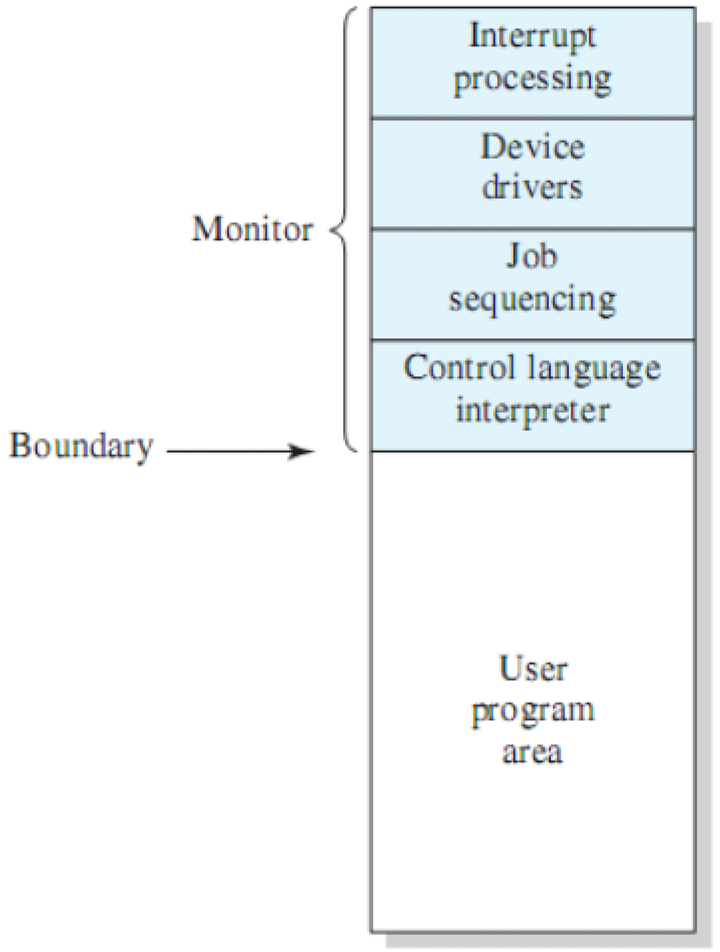


Figure 2.3 **Memory Layout for a Resident Monitor**

PROCESSOR POINT OF VIEW

- ❖Processor executes instruction from the memory containing the monitor
- ❖Executes the instructions in the user program until it encounters an ending or error condition
- ❖"*control is passed to a job*" means processor is fetching and executing instructions in a user program
- ❖"*control is returned to the monitor*" means that the processor is fetching and executing instructions from the monitor program

JOB CONTROL LANGUAGE (JCL)

Special type of programming language used to provide instructions to the monitor

↓

what compiler to use

↓

what data to use

DESIRABLE HARDWARE FEATURES

Memory protection for monitor

- while the user program is executing, it must not alter the memory area containing the monitor

Timer

- prevents a job from monopolizing the system

Privileged instructions

- can only be executed by the monitor

Interrupts

- gives OS more flexibility in controlling user programs

MODES OF OPERATION

User Mode

- user program executes in user mode
- certain areas of memory are protected from user access
- certain instructions may not be executed

Kernel Mode

- monitor executes in kernel mode
- privileged instructions may be executed
- protected areas of memory may be accessed

SIMPLE BATCH SYSTEM OVERHEAD

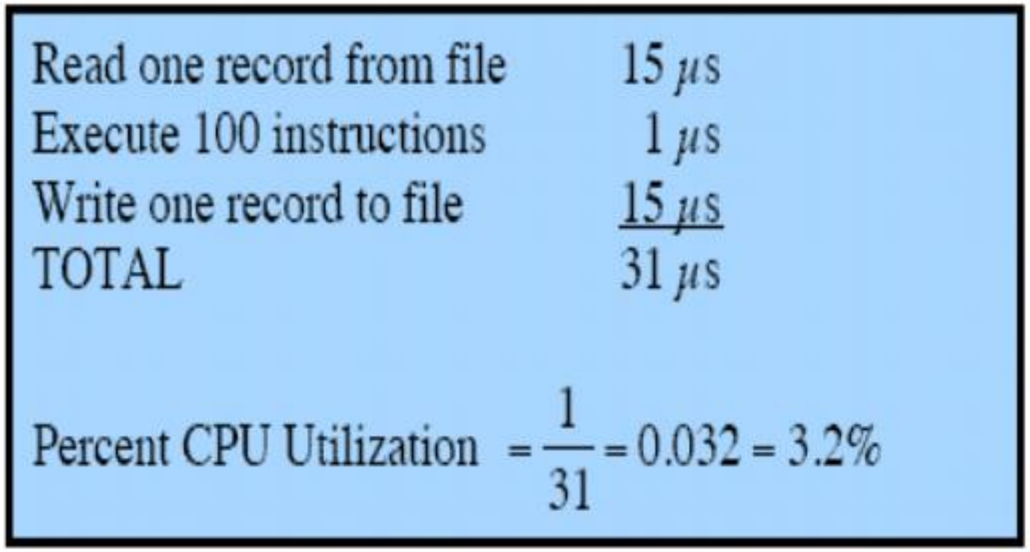
Processor time alternates between execution of user programs and execution of the monitor

Sacrifices:

- Some main memory is now given over to the monitor
- Some processor time is consumed by the monitor

- Despite overhead, the simple batch system improves utilization of the computer

MULTIPROGRAMMED BATCH SYSTEMS



Read one record from file	15 µs
Execute 100 instructions	1 µs
Write one record to file	15 µs
TOTAL	31 µs

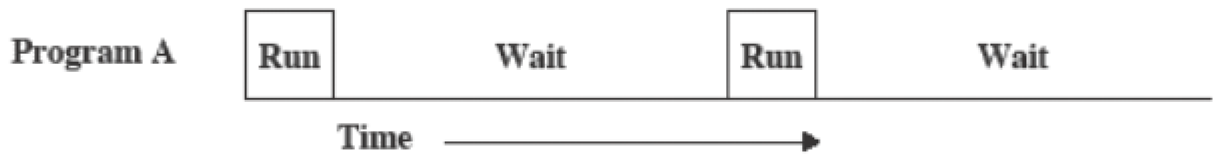
$$\text{Percent CPU Utilization} = \frac{1}{31} = 0.032 = 3.2\%$$

Figure 2.4 System Utilization Example

Processor is often idle

- Even with automatic job sequencing
- I/O devices are slow compared to processor

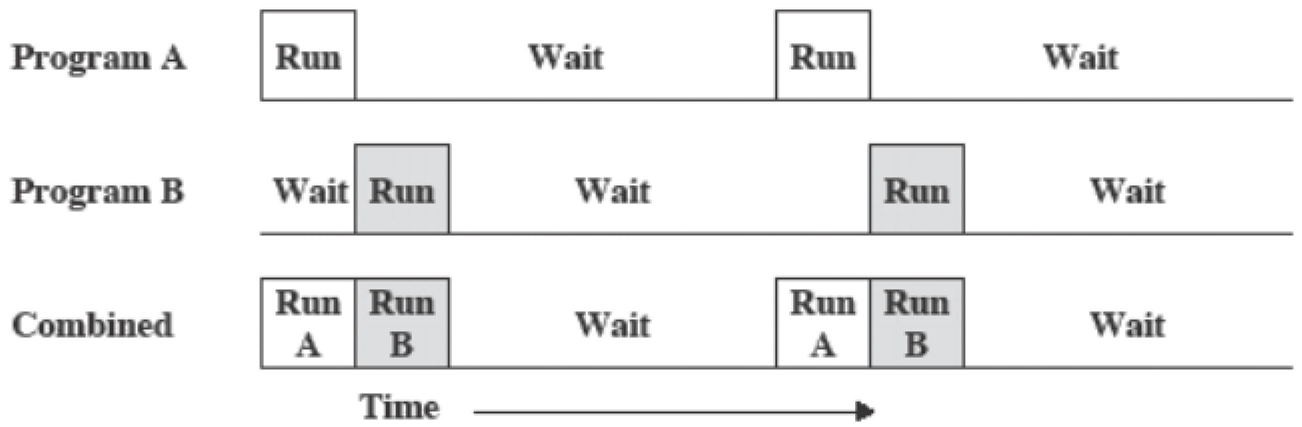
UNIPROGRAMMING



(a) Uniprogramming

The processor spends a certain amount of time executing, until it reaches an I/O instruction; it must then wait until that I/O instruction concludes before proceeding

MULTIPROGRAMMING

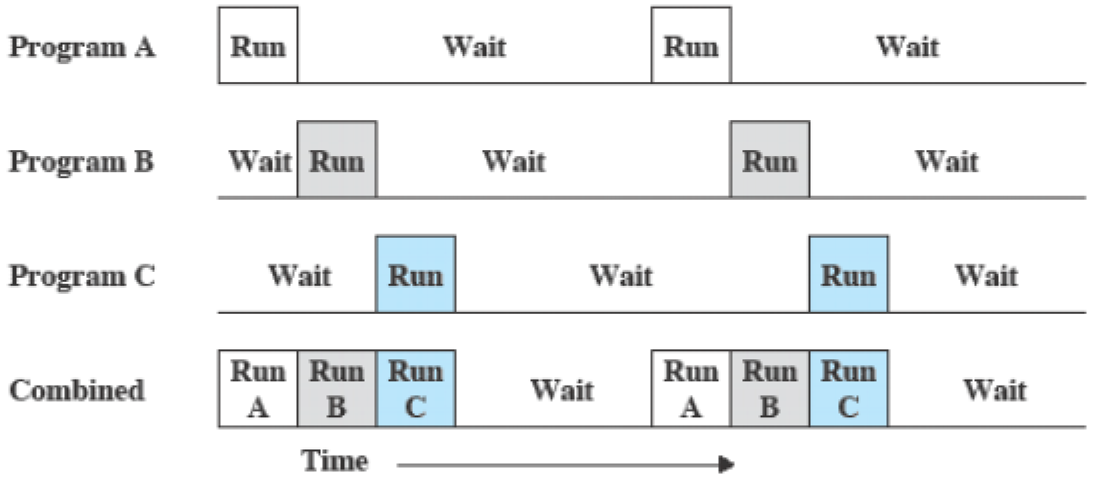


(b) Multiprogramming with two programs

There must be enough memory to hold the OS (resident monitor) and one user program

When one job needs to wait for I/O, the processor can switch to the other job, which is likely not waiting for I/O

MULTIPROGRAMMING



(c) Multiprogramming with three programs

Multiprogramming

- also known as multitasking
- memory is expanded to hold three, four, or more programs and switch among all of them

MULTIPROGRAMMING EXAMPLE

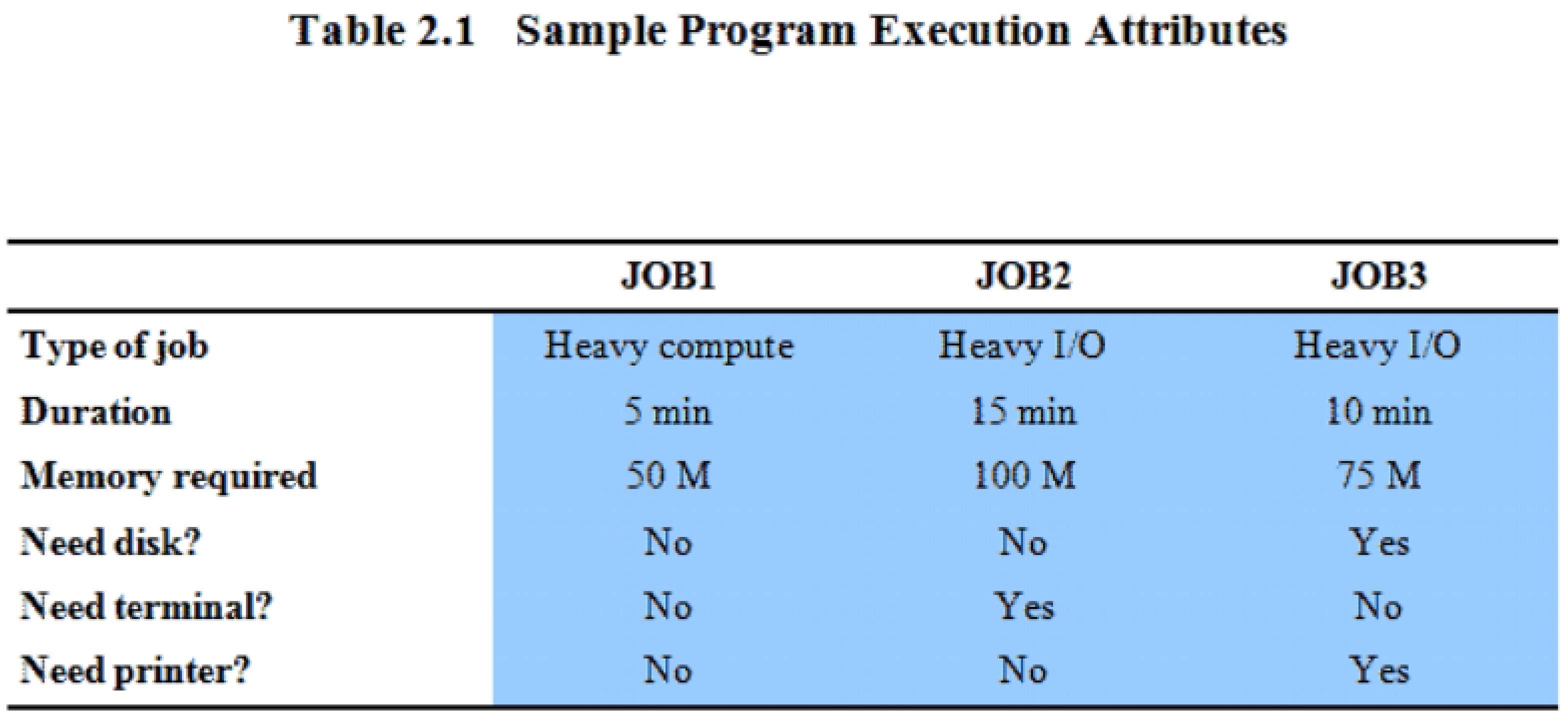


Table 2.1 Sample Program Execution Attributes

	JOB1	JOB2	JOB3
Type of job	Heavy compute	Heavy I/O	Heavy I/O
Duration	5 min	15 min	10 min
Memory required	50 M	100 M	75 M
Need disk?	No	No	Yes
Need terminal?	No	Yes	No
Need printer?	No	No	Yes

EFFECTS ON RESOURCE UTILIZATION

	Uniprogramming	Multiprogramming
Processor use	20%	40%
Memory use	33%	67%
Disk use	33%	67%
Printer use	33%	67%
Elapsed time	30 min	15 min
Throughput	6 jobs/hr	12 jobs/hr
Mean response time	18 min	10 min

Table 2.2 Effects of Multiprogramming on Resource Utilization

UTILIZATION HISTOGRAMS

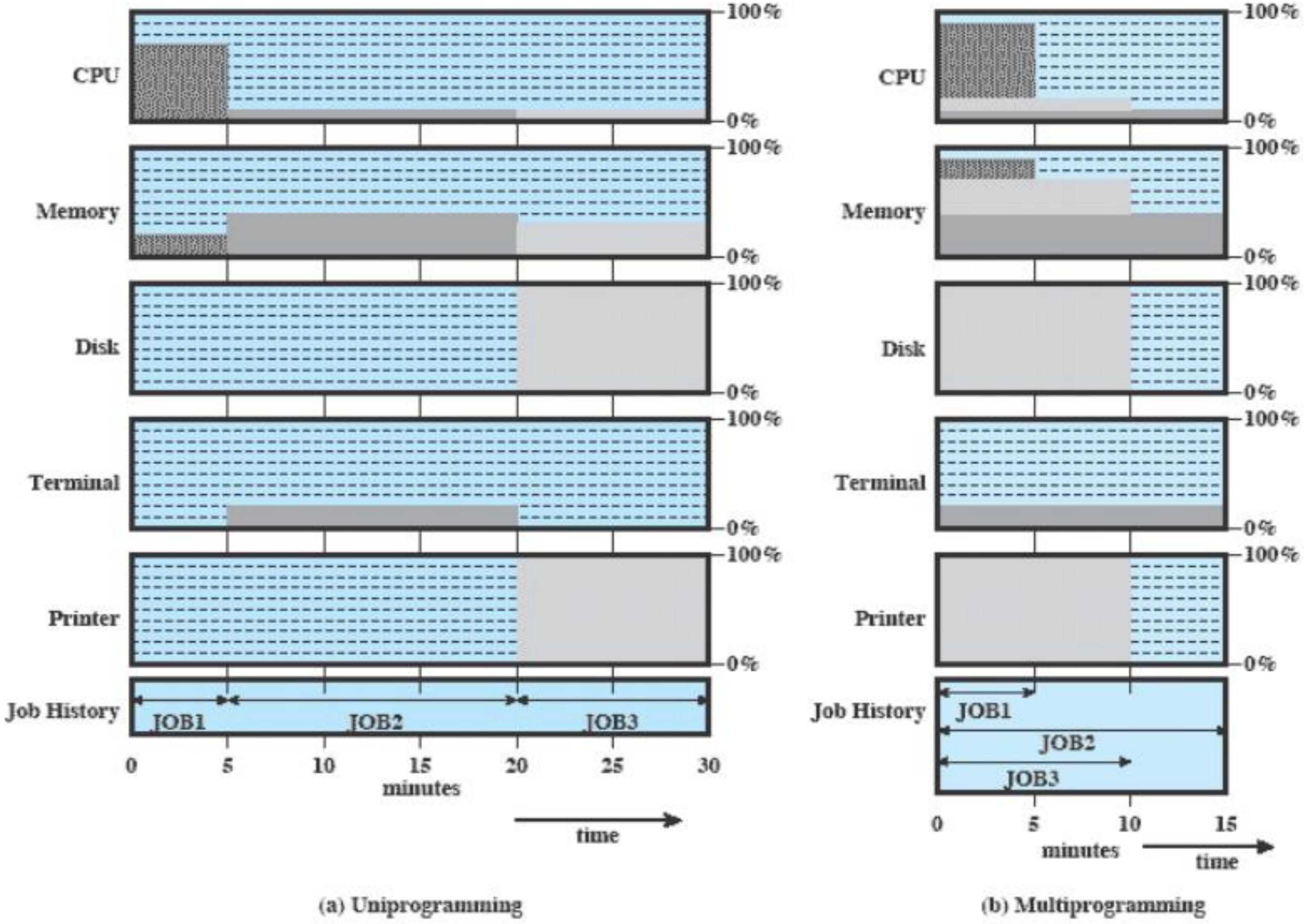
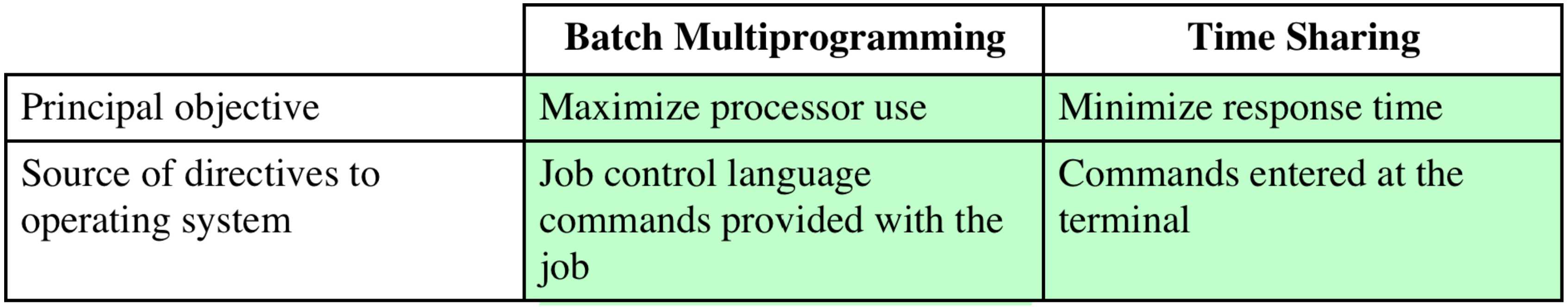


Figure 2.6 Utilization Histograms

TIME-SHARING SYSTEMS

- ❖Can be used to handle multiple interactive jobs
- ❖Processor time is shared among multiple users
- ❖Multiple users simultaneously access the system through terminals, with the OS interleaving the execution of each user program in a short burst or quantum of computation

BATCH MULTIPROGRAMMING VS. TIME SHARING



	Batch Multiprogramming	Time Sharing
Principal objective	Maximize processor use	Minimize response time
Source of directives to operating system	Job control language commands provided with the job	Commands entered at the terminal

Table 2.3 Batch Multiprogramming versus Time Sharing

COMPATIBLE TIME-SHARING SYSTEMS

CTSS

One of the first time-sharing operating systems

- Developed at MIT by a group known as Project MAC
- Ran on a computer with 32,000 36-bit words of main memory, with the resident monitor consuming 5000 words of that memory!
- To simplify both the monitor and memory management a program was always loaded to start at the location of the 5000th word

TIME SLICING

- System clock generates interrupts at a rate of approximately one every 0.2 seconds
- At each interrupt OS regained control and could assign processor to another user
- At regular time intervals the current user would be preempted and another user loaded in
- Old user programs and data were written out to disk
- Old user program code and data were restored in main memory when that program was next given a turn

CTSS OPERATION

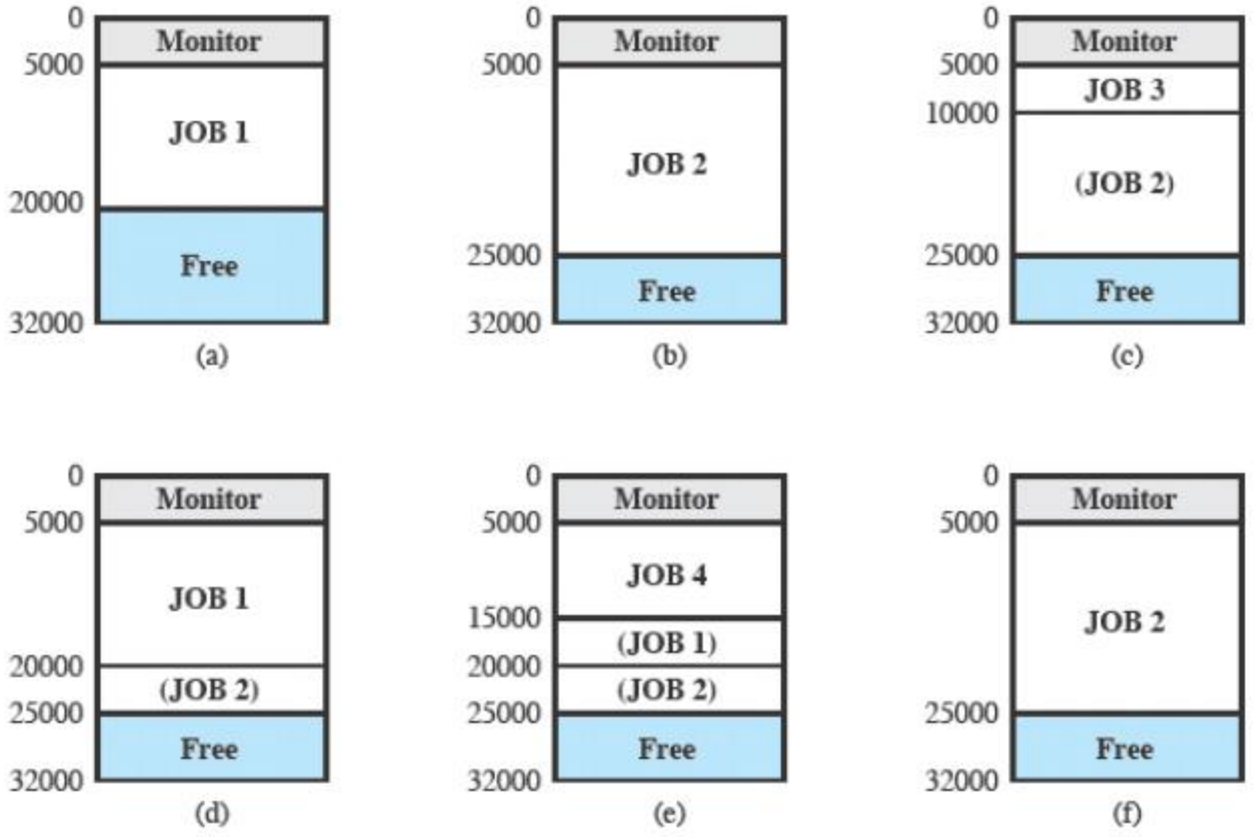


Figure 2.7 CTSS Operation

DIFFERENT ARCHITECTURAL APPROACHES

Demands on operating systems require new ways of organizing the OS

Different Approaches and Design Elements

- Microkernel Architecture
- Multithreading
- Symmetric Multiprocessing
- Distributed Operating Systems
- Object-Oriented Design

MICROKERNEL ARCHITECTURE

Assigns only a few essential functions to the kernel:

- address spaces
- interprocess communication (IPC)
- basic scheduling

- The approach:

- simplifies implementation
- provides flexibility
- is well suited to a distributed environment

MULTITHREADING

Technique in which a process, executing an application, is divided into threads that can run concurrently

Thread

- dispatchable unit of work
- includes a processor context and its own data area to enable subroutine branching
- executes sequentially and is interruptible

Process

- a collection of one or more threads and associated system resources
- programmer has greater control over the modularity of the application and the timing of application related events

SYMMETRIC MULTIPROCESSING (SMP)

- ❖Term that refers to a computer hardware architecture and also to the OS behavior that exploits that architecture
- ❖Several processes can run in parallel
- ❖Multiple processors are transparent to the user
  - these processors share same main memory and I/O facilities
  - all processors can perform the same functions
- ❖The OS takes care of scheduling of threads or processes on individual processors and of synchronization among processors

SMP ADVANTAGES

Performance	more than one process can be running simultaneously, each on a different processor
Availability	failure of a single process does not halt the system
Incremental Growth	performance of a system can be enhanced by adding an additional processor
Scaling	vendors can offer a range of products based on the number of processors configured in the system

MULTIPROGRAMMING

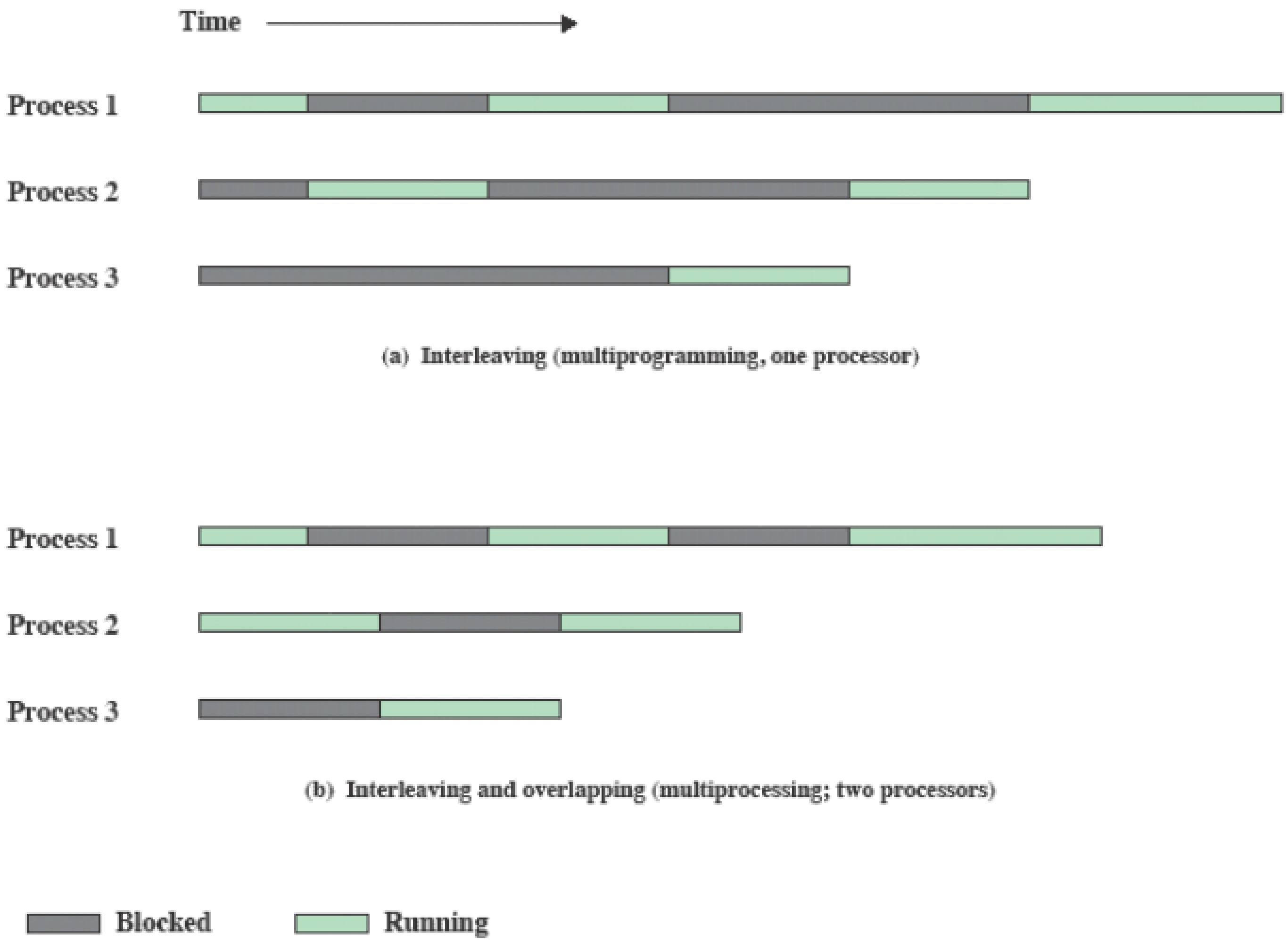

Time
Process 1
Process 2
Process 3
(a) Interleaving (multiprogramming, one processor)
Process 1
Process 2
Process 3
(b) Interleaving and overlapping (multiprocessing; two processors)
Blocked
Running

Figure 2.12 Multiprogramming and Multiprocessing

VIRTUAL MACHINES AND VIRTUALIZATION

Virtualization

- Enables a single PC or server to simultaneously run multiple operating systems or multiple sessions of a single OS
- A machine can host numerous applications, including those that run on different operating systems, on a single platform
- Host operating system can support a number of virtual machines (VM)
  - each has the characteristics of a particular OS and, in some versions of virtualization, the characteristics of a particular hardware platform.

VIRTUAL MEMORY CONCEPT

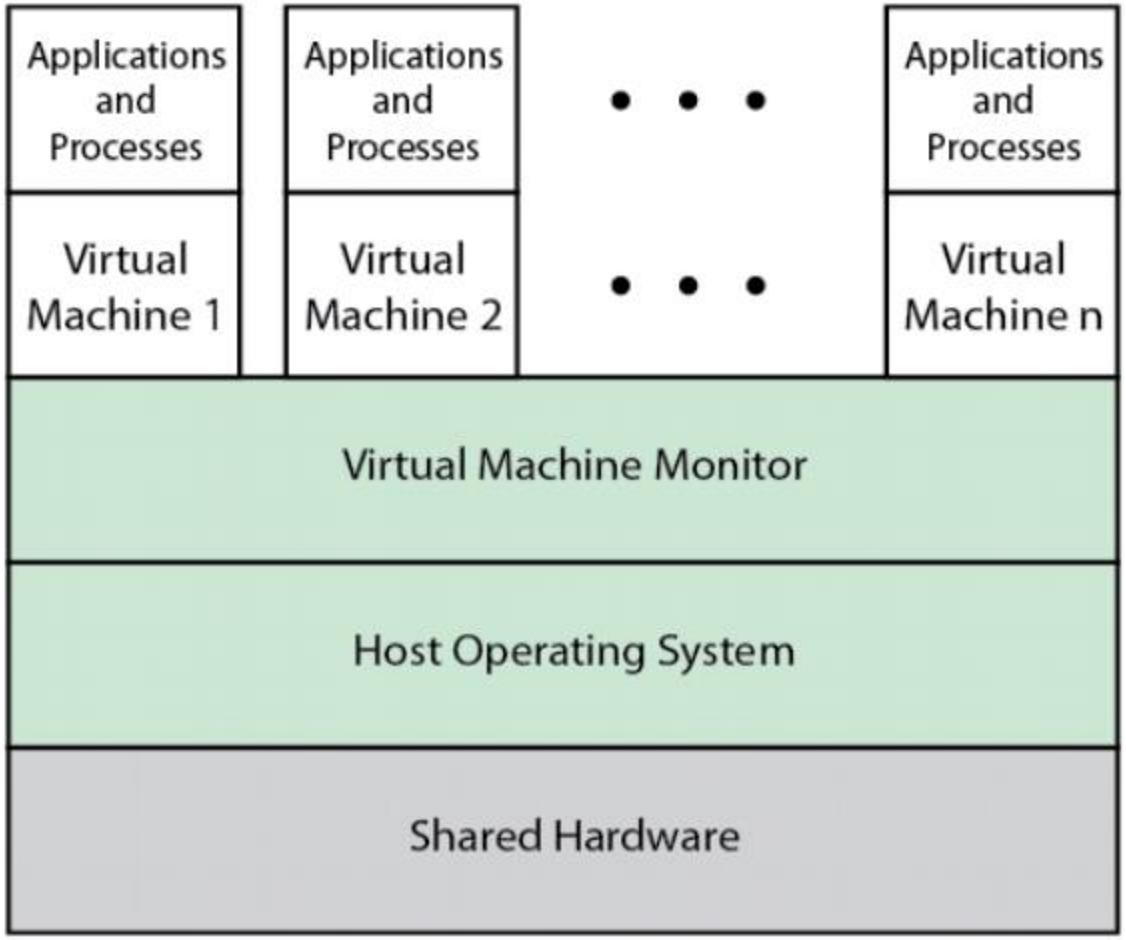


Figure 2.13 Virtual Memory Concept

VIRTUAL MACHINE ARCHITECTURE

Process perspective:

- the machine on which it executes consists of the virtual memory space assigned to the process
- the processor registers it may use
- the user-level machine instructions it may execute
- OS system calls it may invoke for I/O
- ABI defines the machine as seen by a process

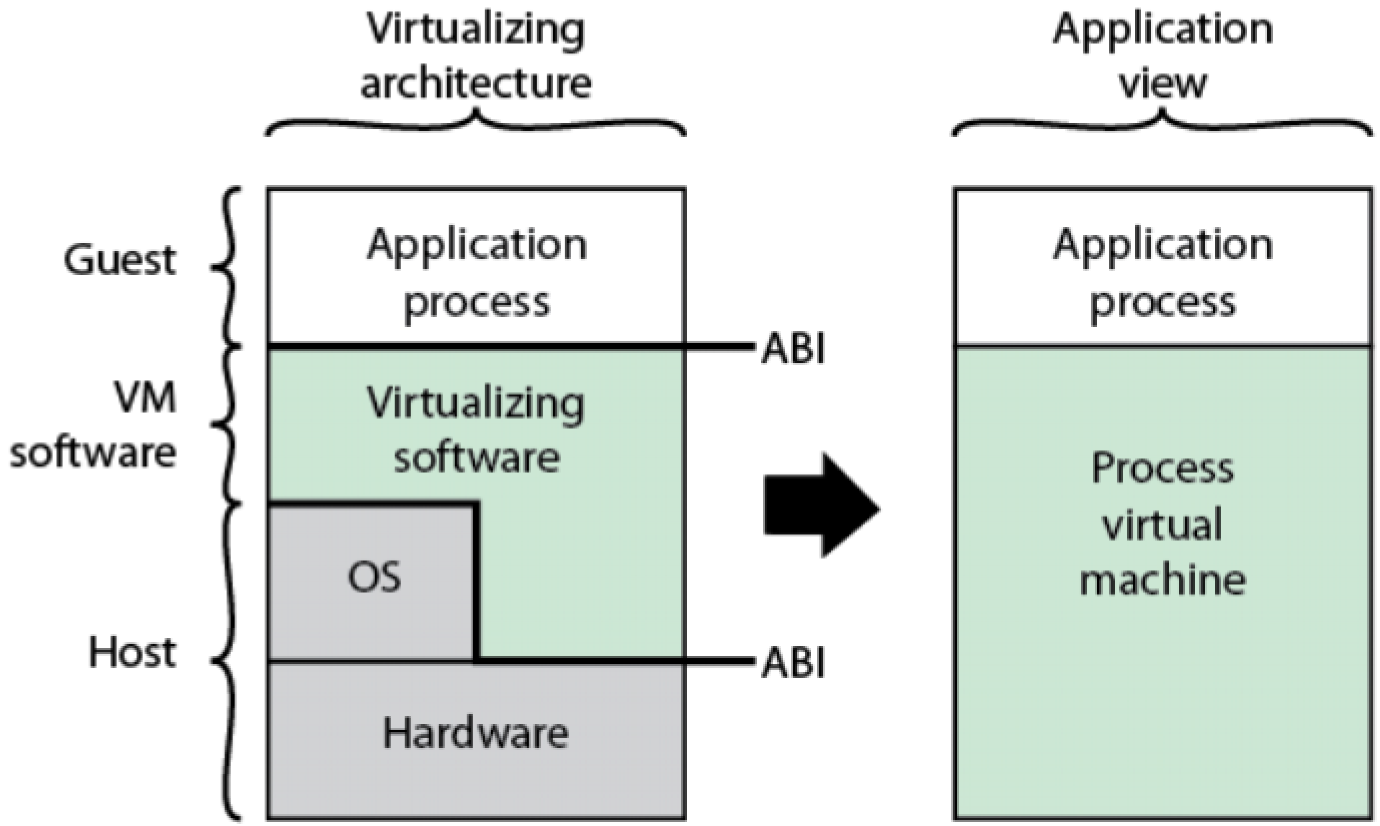
Application perspective:

- machine characteristics are specified by high-level language capabilities and OS system library calls
- API defines the machine for an application

OS perspective:

- processes share a file system and other I/O resources
- system allocates real memory and I/O resources to the processes
- ISA provides the interface between the system and machine

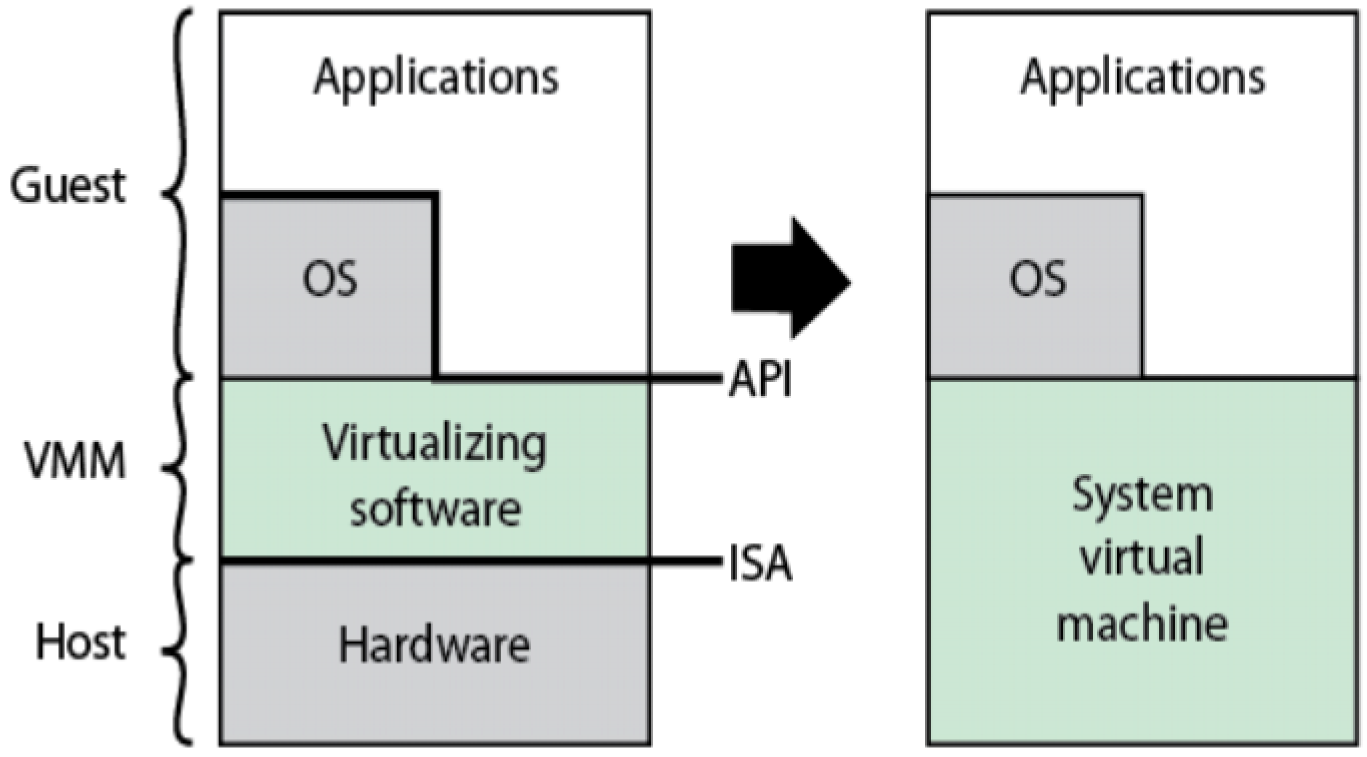
PROCESS AND SYSTEM VIRTUAL MACHINES



(a) Process VM

Figure 2.14 Process and System Virtual Machines

PROCESS AND SYSTEM VIRTUAL MACHINES



(b) System VM

Figure 2.14 Process and System Virtual Machines

SYMMETRIC MULTIPROCESSOR OS CONSIDERATIONS

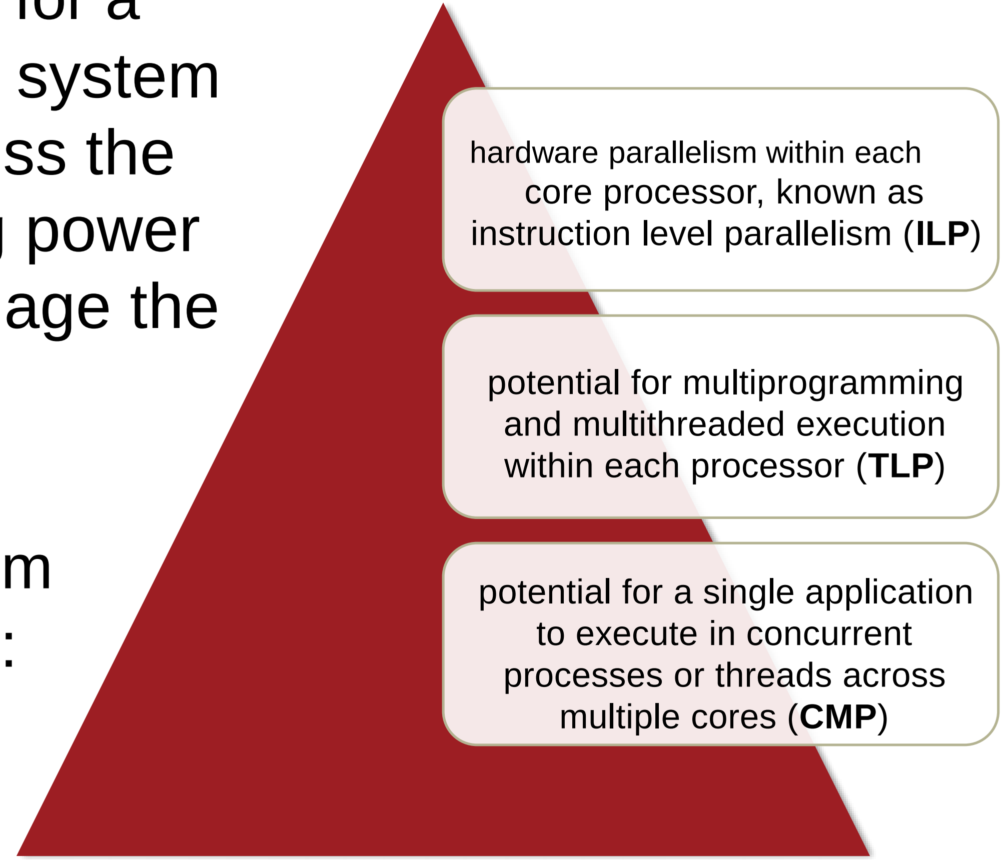
A multiprocessor OS must provide all the functionality of a multiprogramming system plus additional features to accommodate multiple processors

Key design issues:

- **Simultaneous concurrent processes or threads**
  - kernel routines need to be reentrant to allow several processors to execute the same kernel code simultaneously
- **Scheduling**
  - any processor may perform scheduling, which complicates the task of enforcing a scheduling policy
- **Synchronization**
  - with multiple active processes having potential access to shared address spaces or shared I/O resources, care must be taken to provide effective synchronizatio
- **Memory Management**
  - the reuse of physical pages is the biggest problem of concern
- **Reliability and Fault Tolerance**
  - the OS should provide graceful degradation in the face of processor failure

MULTICORE OS CONSIDERATIONS

- The design challenge for a many-core multicore system is to efficiently harness the multicore processing power and intelligently manage the substantial on-chip resources efficiently
- Potential for parallelism exists at three levels:


hardware parallelism within each core processor, known as instruction level parallelism (**ILP**)
potential for multiprogramming and multithreaded execution within each processor (**TLP**)
potential for a single application to execute in concurrent processes or threads across multiple cores (**CMP**)

GENERAL UNIX ARCHITECTURE

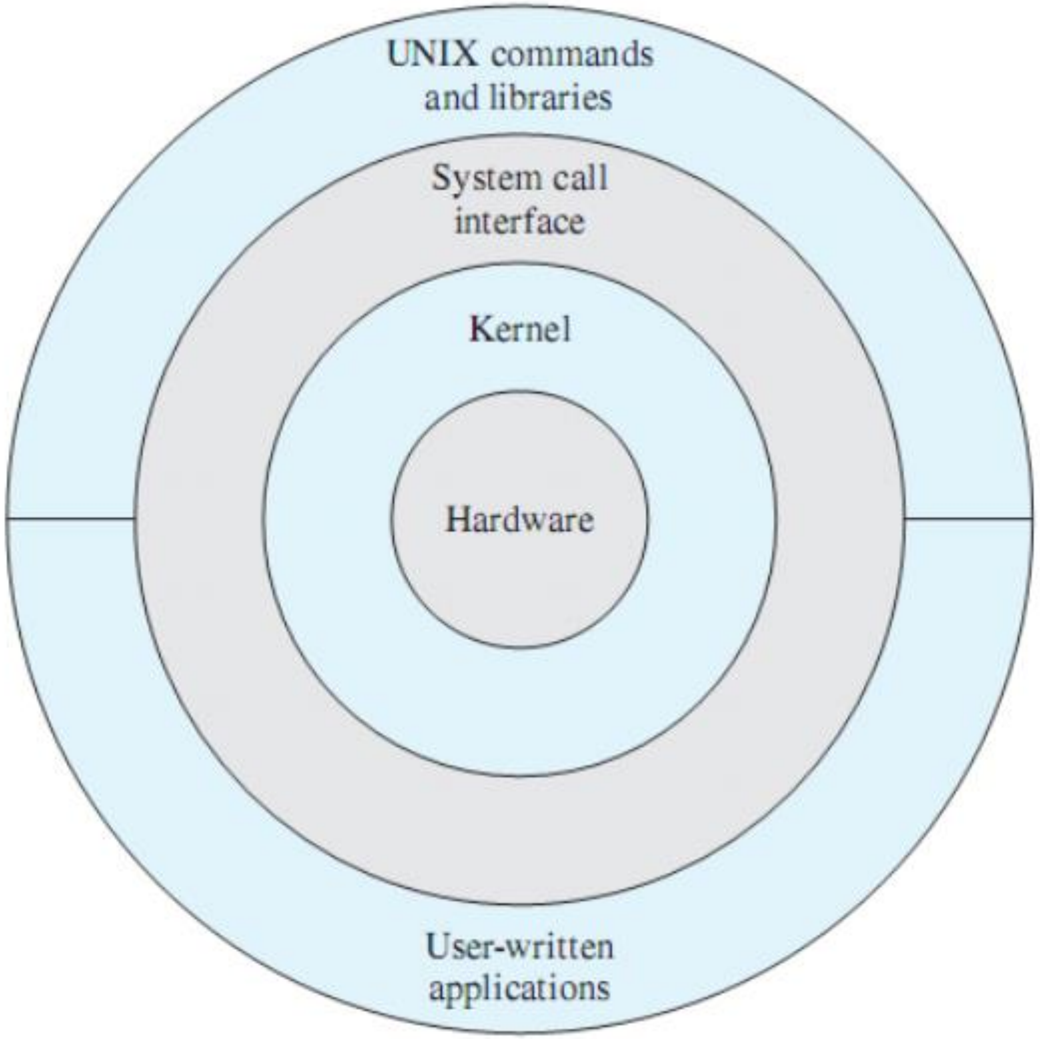


Figure 2.14 **General UNIX Architecture**

MODULAR MONOLITHIC KERNEL

- Includes virtually all of the OS functionality in one large block of code that runs as a single process with a single address space
- All the functional components of the kernel have access to all of its internal data structures and routines
- Linux is structured as a collection of modules

LOADABLE MODULES

- Relatively independent blocks
- A module is an object file whose code can be linked to and unlinked from the kernel at runtime
- A module is executed in kernel mode on behalf of the current process
- Have two important characteristics:
  - Dynamic linking
  - Stackable modules

KERNEL AND SHELL

Unix-like systems divide the OS into

- **Kernel**
    - The lowest part of the OS that **talks to the physical hardware**.
    - Implements **Process/Memory Management** etc.
    - Runs in **supervisor mode**.
- **Shell**
    - Accepts commands from the user.
    - Shells for Unix-like systems allow **combining simple programs to achieve a complex task**.
    - Runs in user mode.

SYSTEM CALLS

System calls are the mechanism through which services of the operating systems are sought.

Examples

- Starting a new process or thread
- Reading contents of a file
- Existing a program

SYSTEM CALLS

- ❖A system call starts with C/C++ procedure call
- ❖The procedure store the *call number* at some special place and executes a trap instruction.
- ❖The system enters kernel mode and starts execution from a fixed memory location as per the *call number*.
- ❖After performing the task in kernel mode the system returns to user mode and transfers control back to the user program.

SYSTEM CALLS FOR PROCESS MANAGEMENT

Call	Description
`pid = fork()`	Create a child process identical to the parent
`pid = waitpid(pid, &statloc, options)`	Wait for a child to terminate
`s = execve(name, argv, environp)`	Replace a process' core image
`exit(status)`	Terminate process execution and return status

SYSTEM CALLS FOR FILE MANAGEMENT

Call	Description
`fd = open(fife, how,...)`	Open a file for reading, writing, or both
`s = close(fd)`	Close an open file
`n = read(fd, buffer, nbytes)`	Read data from a file into a buffer
`n = write(fd, buffer, nbytes)`	Write data from a buffer into a file
`position = lseek(fd, offset, whence)`	Move the file pointer
`s = stat(narne, &buf)`	Get a fife's status information

SYSTEM CALLS FOR DIRECTORY MANAGEMENT

Call	Description
`s = mkdir(name, mode)`	Create a new directory
`s = rmdir(name)`	Remove an empty directory
`s = link(name1, name2)`	Create a new entry, name2, pointing to namel
`s = unlink(name)`	Remove a directory entry
`s = mount(speciaf, name, flag)`	Mount a file system
`s = umount(special)`	Unmount a file system

MISCELLANEOUS SYSTEM CALLS

Call	Description
`s = chdir(dirname)`	Change the working directory
`s = chmod(name, mode)`	Change a file's protection bits
`s = kill(pid, signal)`	Send a signal to a process
`seconds = time(&seconds)`	Get the elapsed time since Jan. 1, 1970