SLIIT

Discover Your Future

[redacted]

IT2060/IE2061

Operating Systems and System Administration

Lecture 01

Introduction to Operating System

SLIIT
FACULTY OF COMPUTING

Introduction

[redacted]

- Learn the major components of the operating systems

- Practice with utilities of Unix system administration.

- Students will also apply the knowledge they learn in the lectures, tutorial and labs to complete the unit's programming assignment

[redacted]

Method of Delivery

- 2 Hours Lecture
- 1 Hour Tutorial
- 2 Hours Practical
- Attendance will be marked.

Assessment Criteria

[redacted]

- Mid Term Test (Online) 20% Lessons 1 to 5

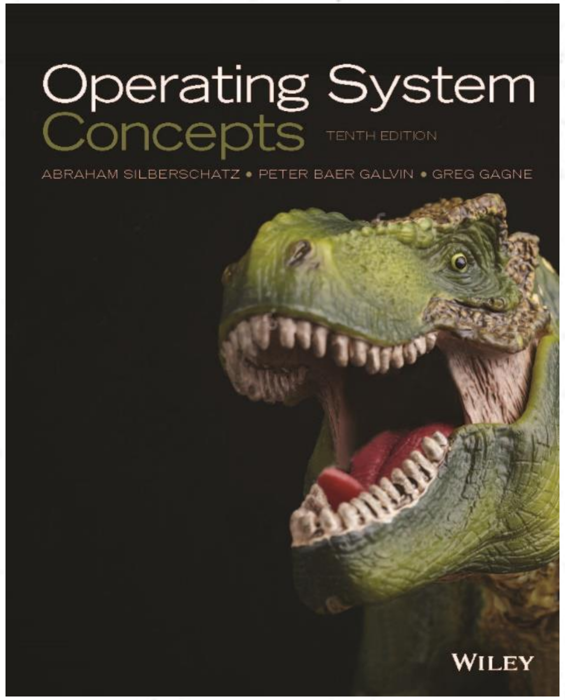
- Assignment (Online) 20% Based on Practical Sessions (C Language)

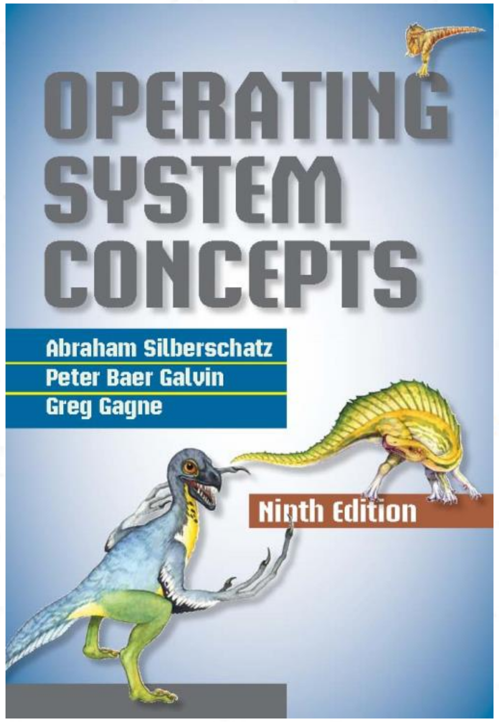
- Final Examination (Written) 60% Lessons 6 to 12

References

[redacted]

A. Silberschatz, P.B. Galvin, G. Gagne, Operating System Concepts, 10th Edition, John Wiley & Sons, 2018


Operating System Concepts
TENTH EDITION
ABRAHAM SILBERSCHATZ • PETER BAER GALVIN • GREG GAGNE
WILEY

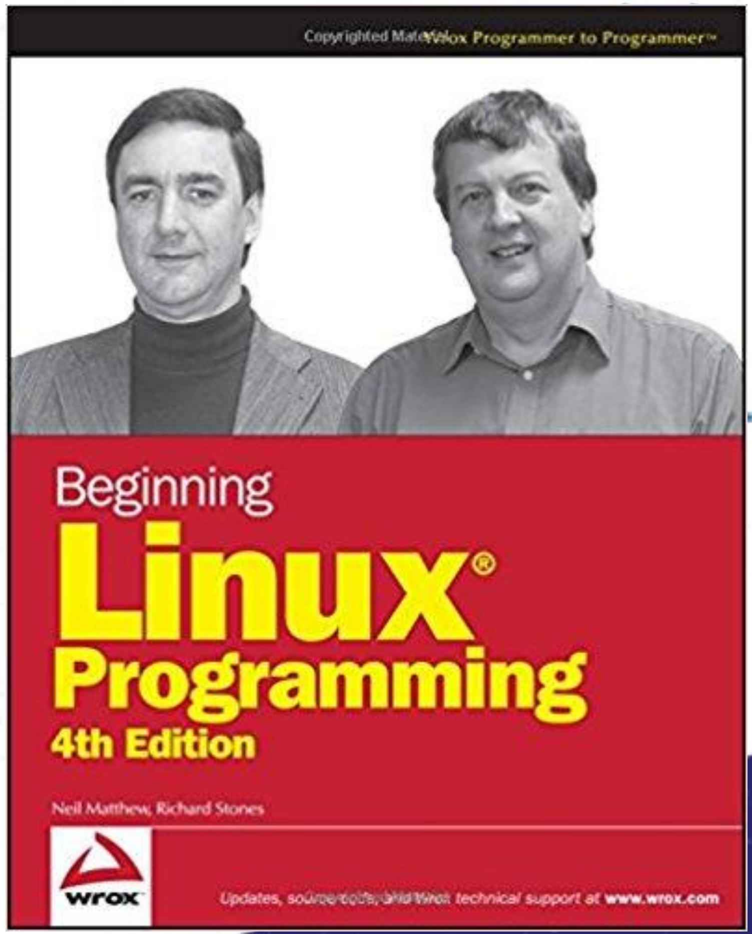


Linux Programming

[redacted]

Beginning Linux Programming
4th Edition
by Neil Matthew Richard Stones

PThreads Primer
A Guide to Multithreaded Programming
Bil Lewis Daniel J. Berg


Wrox Programmer to Programmer
Beginning
Linux
Programming
4th Edition
Neil Matthew, Richard Stones
wrox
Updates, source code, and Wrox technical support at www.wrox.com

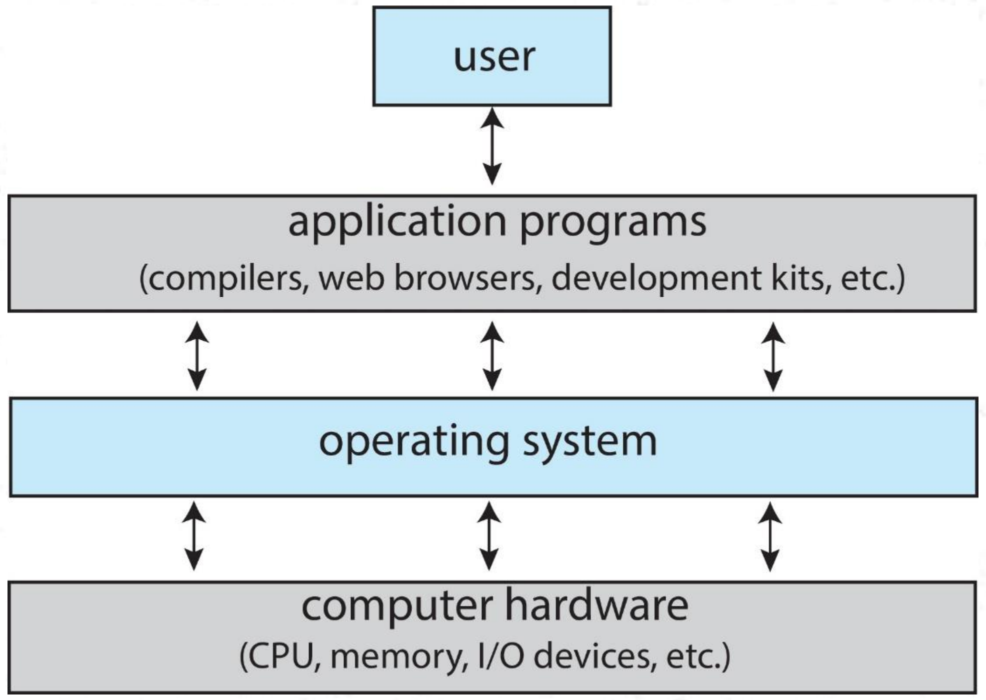
Computer System Structure

[redacted]

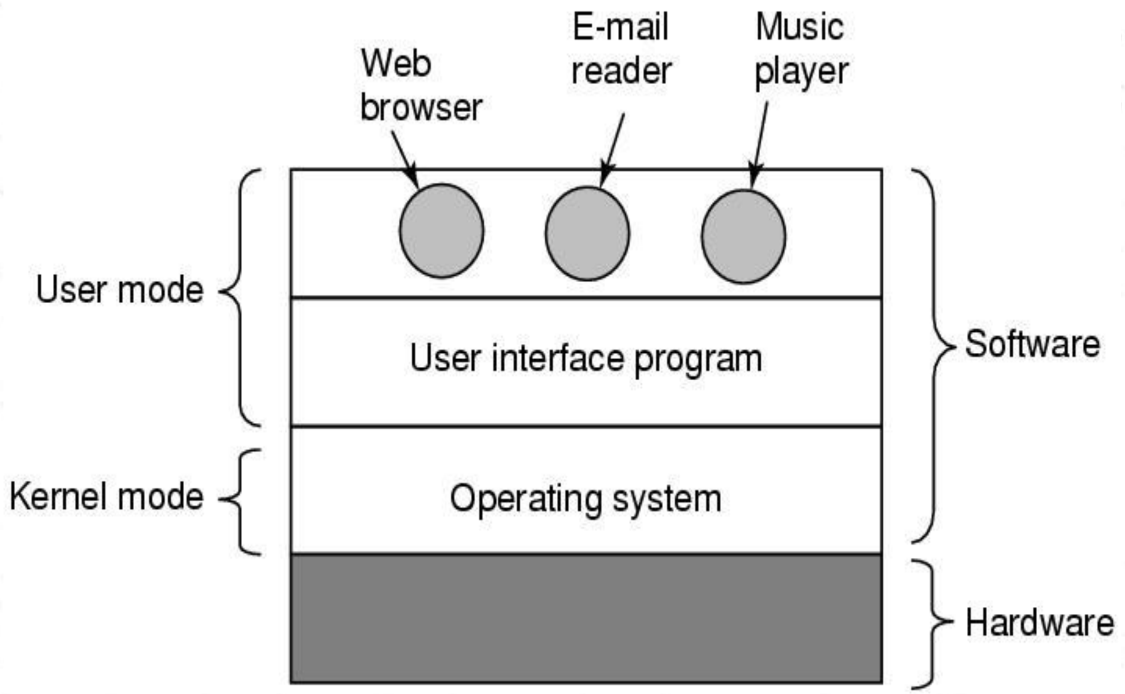
- Computer system can be divided into four components:
  - Hardware – provides basic computing resources
    - CPU, memory, I/O devices
  - Operating system
    - Controls and coordinates use of hardware among various applications and users
  - Application programs – define the ways in which the system resources are used to solve the computing problems of the users
    - Word processors, compilers, web browsers, database systems, video games
  - Users
    - People, machines, other computers

[redacted]

Abstract View of Components of Computer



user
application programs
(compilers, web browsers, development kits, etc.)
operating system
computer hardware
(CPU, memory, I/O devices, etc.)



What is an Operating System?

[redacted]

- An Operating System is a program that acts as an intermediary/interface between a user of a computer and the computer hardware.
- OS goals:
  - Control/execute user/application programs.
  - Make the computer system convenient to use.
  - Ease the solving of user problems.
  - Use the computer hardware in an efficient manner.

Purposes of OS:

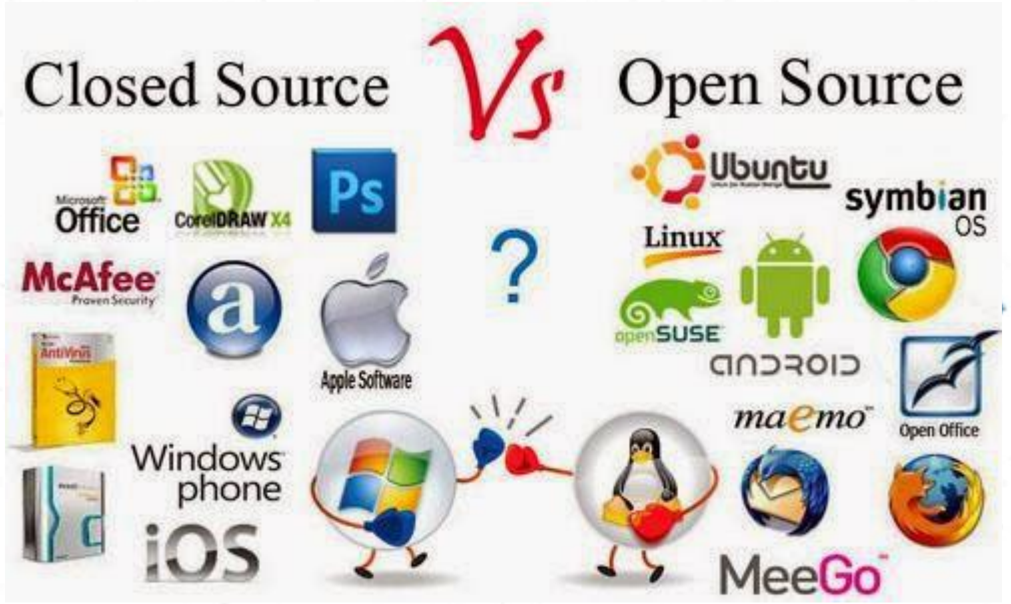
[redacted]

- Provide the environment for program execution and development
- Manage the resources (CPU, memory, IO devices, hard disk, files etc..)
- Provide the access controlling (username and password)

[redacted]

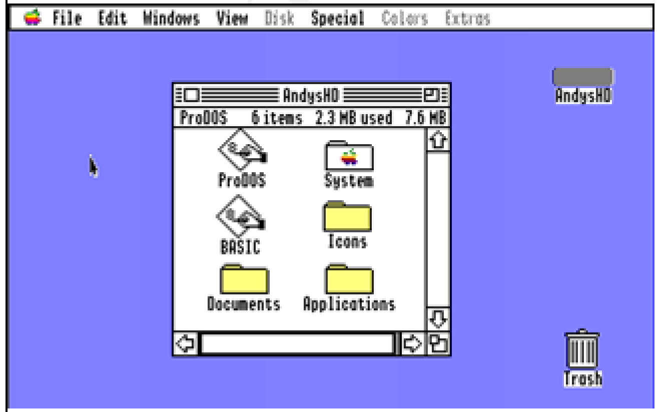
Types of OS

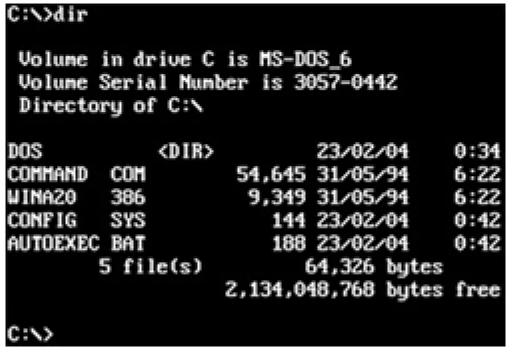
Open source OS	Proprietary OS
Freely downloadable, code can be opened	Buy the OS and code is locked
Unix, Linux, Minix, Fedora, Ubuntu	MS-Windows , Mac OS ,DOS



User Interface of the Operating Syste [redacted]

Graphical User Interface (GUI)	Command Line Interface (CLI)
Very user-friendly and easy to learn the OS	Not much user-friendly and user has to remember the commands
Slow and need more resources	Fast and need less resources
Main components are icons, menu, windows and pointer	Main component is command interpreter





```
C:\>dir

 Volume in drive C is MS-DOS_6
 Volume Serial Number is 3057-0442
 Directory of C:\

DOS          <DIR>        23/02/04    0:34
COMMAND  COM       54,645 31/05/94    6:22
WINA20   386        9,349 31/05/94    6:22
CONFIG   SYS          144 23/02/04    0:42
AUTOEXEC BAT          188 23/02/04    0:42
        5 file(s)        64,326 bytes
                  2,134,048,768 bytes free

C:\>
```

[redacted]

OS Components

- Process Management
- Main-memory management
- Secondary-storage management
- File Management
- I/O System Management
- Protection System
- Networking (Distributed Systems)
- Command- interpreter System

OS Services

[redacted]

For helping users:

- Provide user interface (UI)
  - Command line interface – using text commands
  - Batch interface – commands and their directives are put in a file
  - Graphical user interface (GUI) – window system with pointing devices
- Provide environment for program execution.
  - OS must load program and run it.
  - Program must end normally or abnormally.
- Provide some means to do I/O
  - user programs cannot execute I/O operations directly.
- Provide mechanism to do file-system manipulation.
  - Capability to read, write, create, and delete files, directory trees etc.
- Provide mechanism for process communication.
  - Exchange information between processes executing on the same computer or on different systems through a network.
  - Implemented via shared memory or message passing.
- Detect errors and take appropriate actions to ensure correct and consistent computing.
  - Detect errors in CPU and memory hardware, in I/O devices, or in user programs.

OS Services

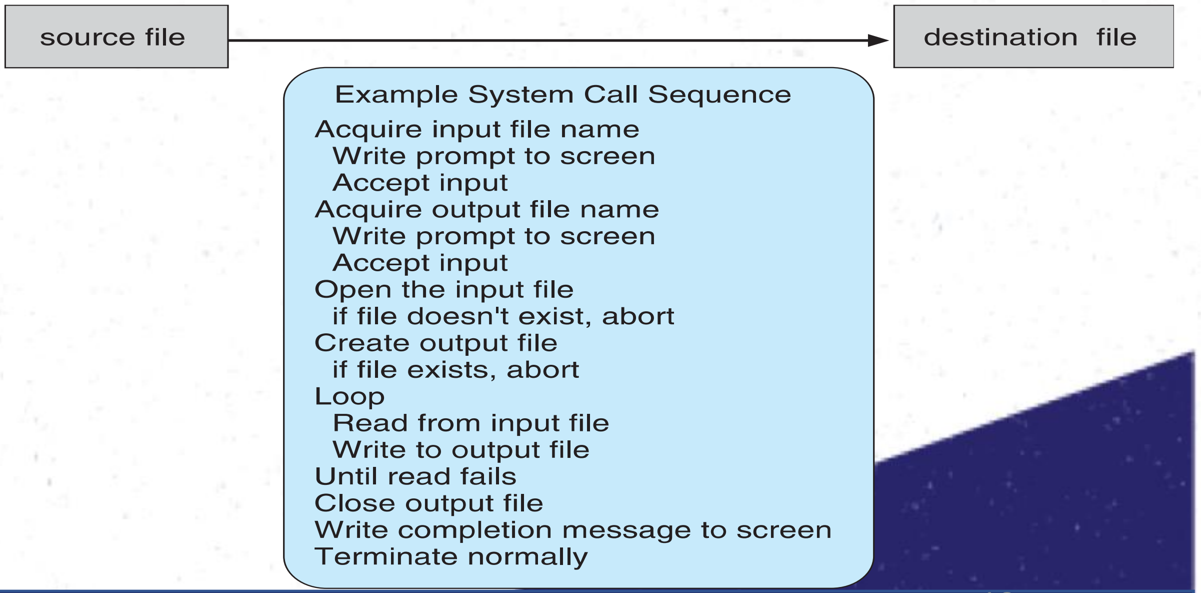
For efficient operation:

- Resource allocation
    - Allocating resources to multiple users or multiple jobs running at the same time (CPU scheduling, etc.).
- Accounting
    - Keep track of and record which users use how much and what kinds of computer resources for account billing or for accumulating usage statistics .
- Protection and security
    - Ensuring that all access to system resources is controlled (access permissions, etc.).

[redacted]

System Calls

- A system call is a user interface to the OS services
  - Available in assembly-language instructions or in high level languages for systems programming (e.g., C)
    - E.g., fork () is a system call to ask OS to create a new process
  - Application Program Interface (API): a set of functions available to an application programmer.
    - An API typically invokes the actual system calls for the application programmers.
    - Examples of API: Windows API, POSIX API, Java API
    - API is more portable and simpler to use

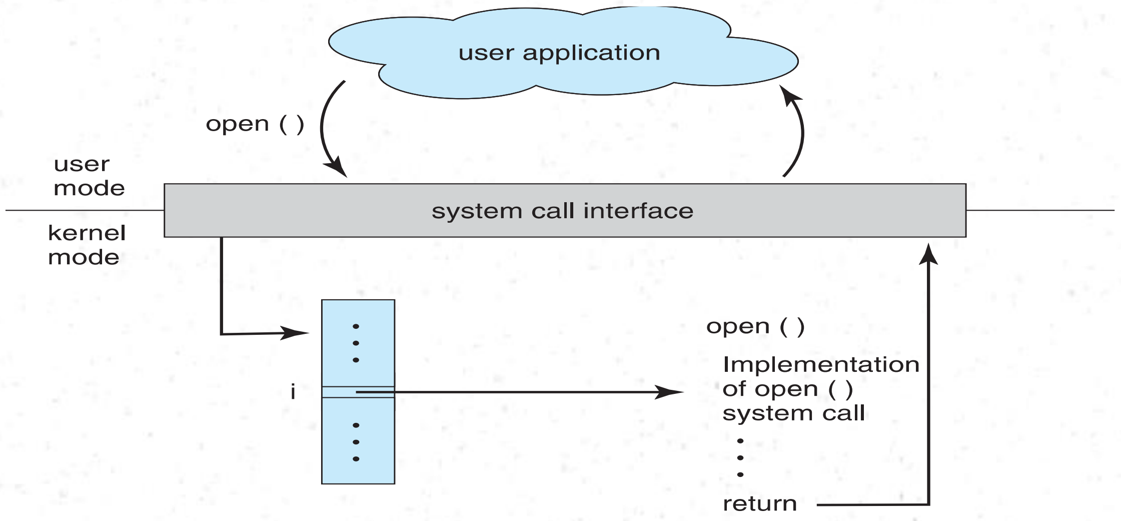


Example System Call Sequence

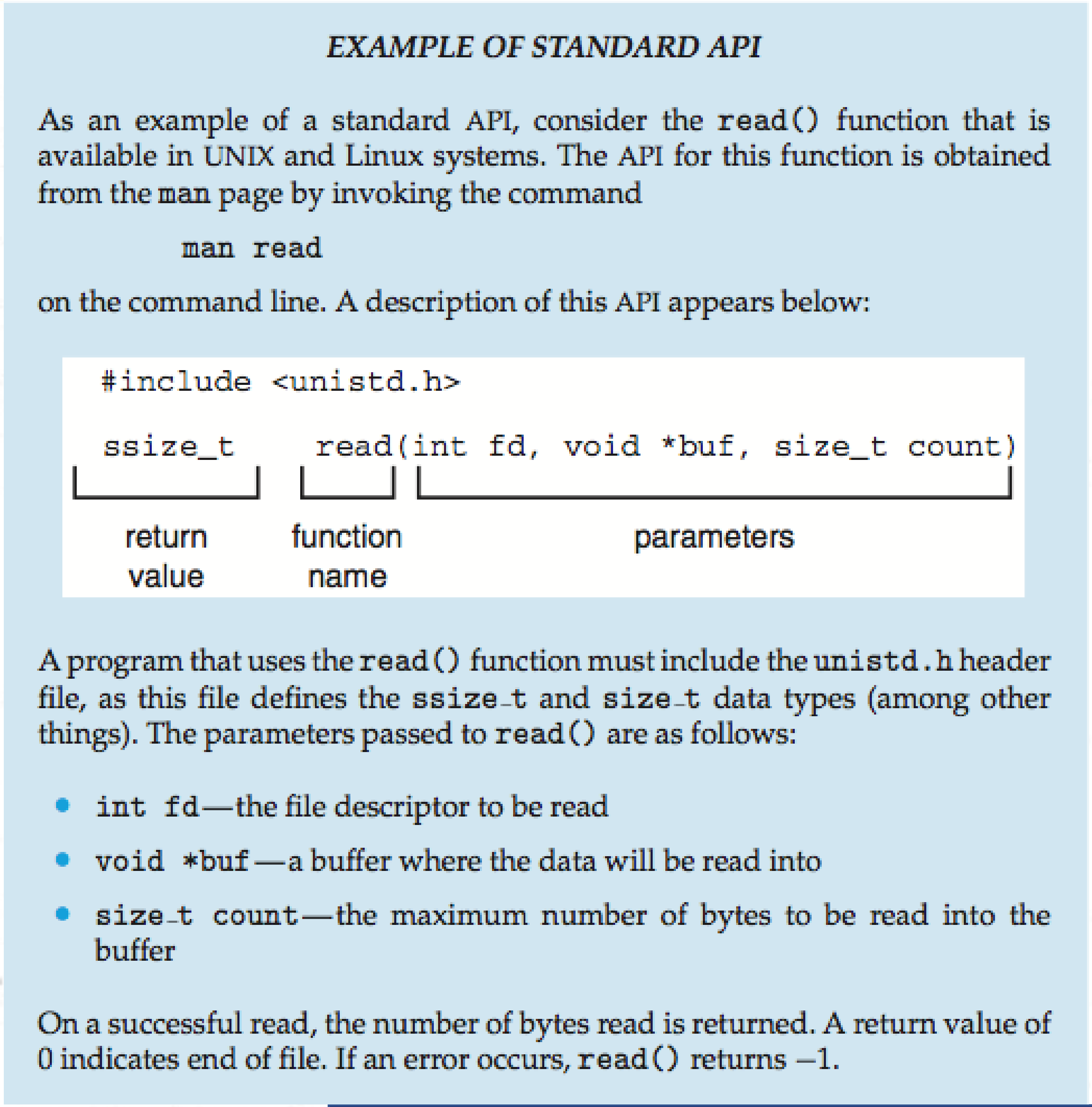
```
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close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

System Calls (contd.)

- System call result/termination.
  - Normal termination (exit or return).
  - Terminate current program and return to the command interpreter.
  - Abnormal termination (trap) — program error.



[redacted]

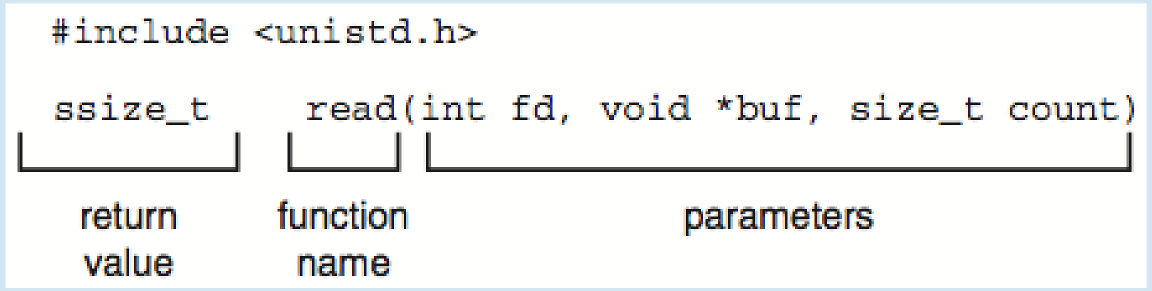


EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:



```
#include <unistd.h>

ssize_t    read(int fd, void *buf, size_t count)
```

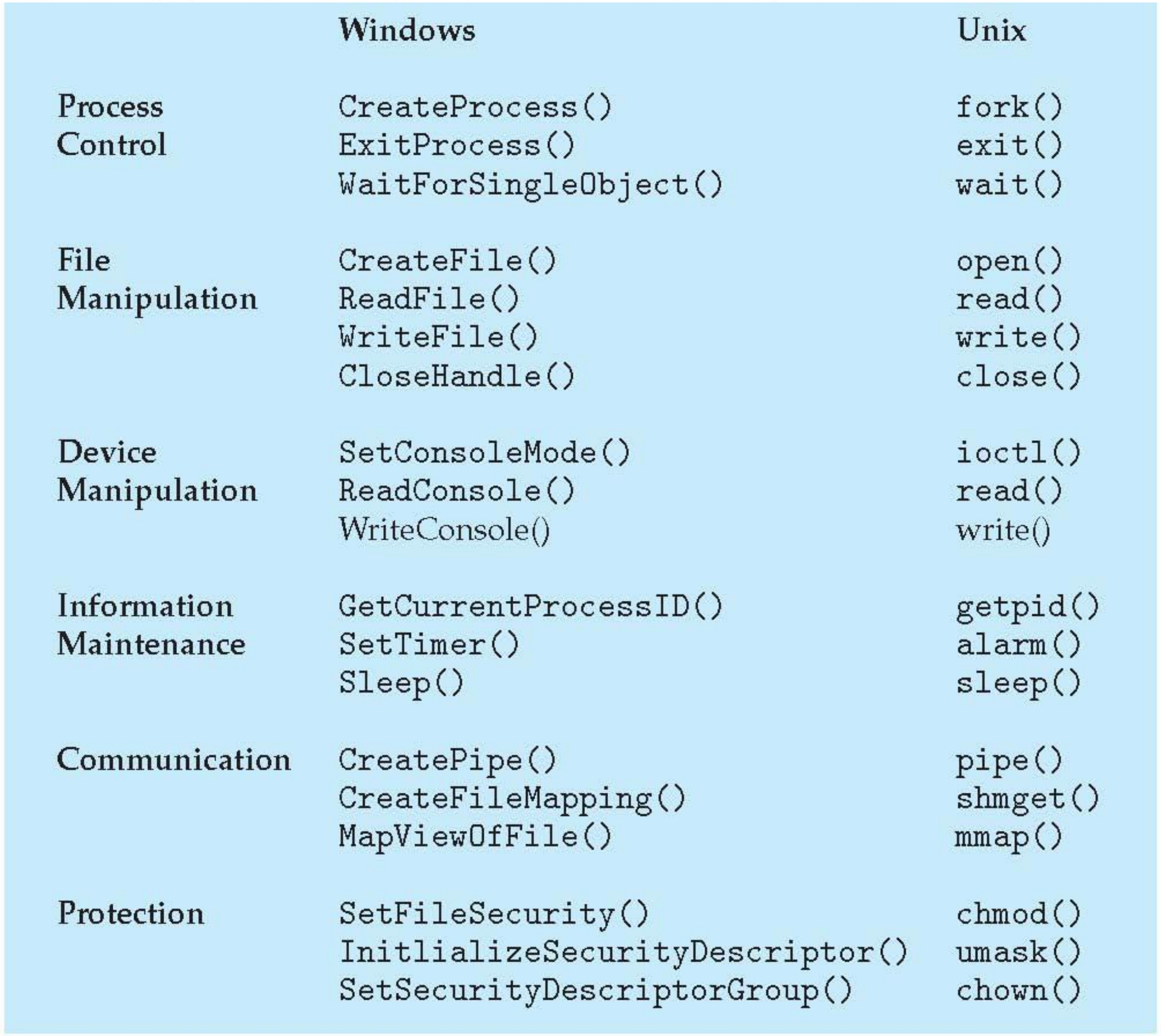
A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

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

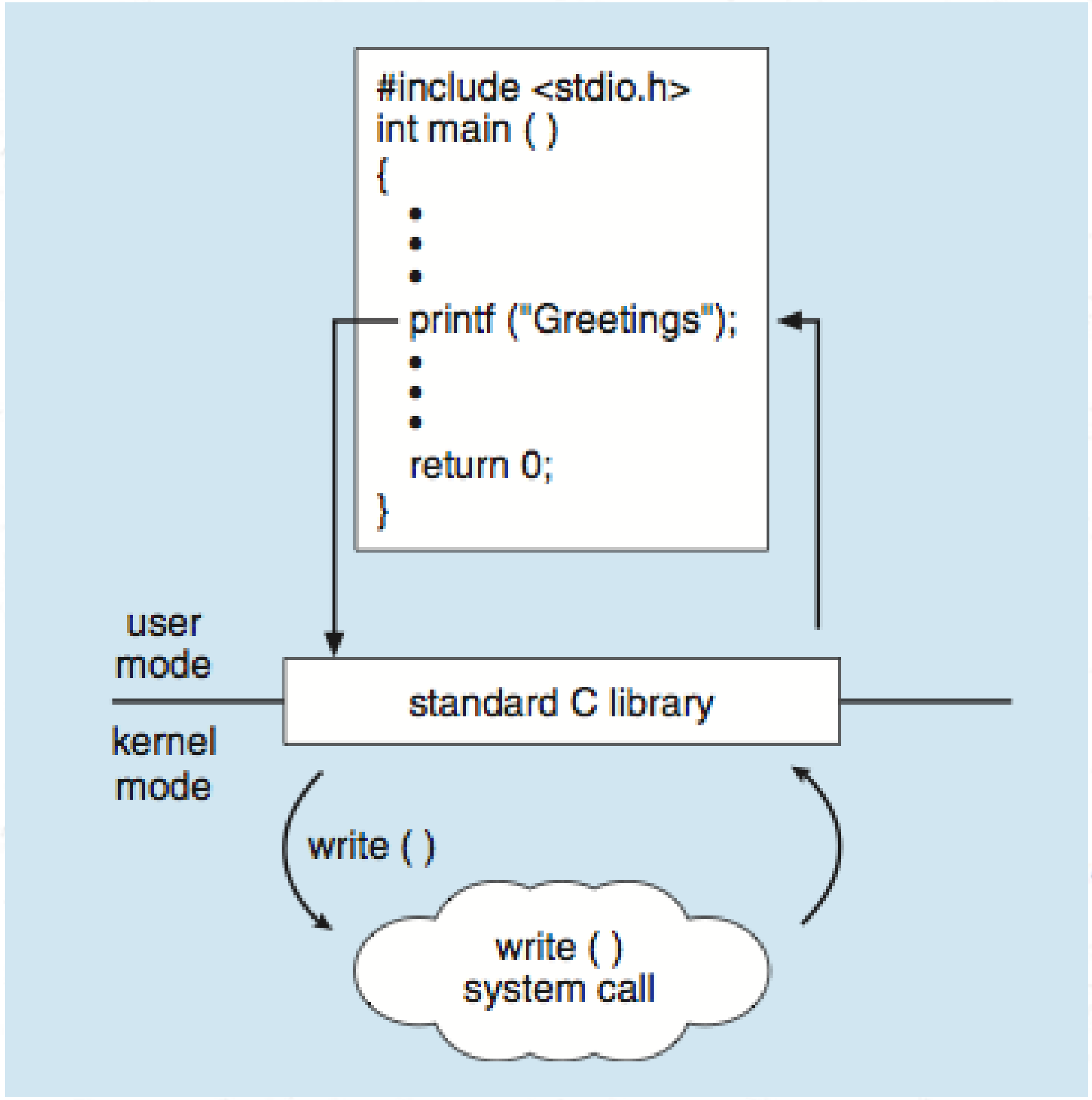
SLIIT FACULTY OF COMPUTING

[redacted]



	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()

[redacted]



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

System Calls (cont.)

[redacted]

Types of system calls

1) Process control
   - End, abort (running program); Load, execute; Create, terminate; Signal event, etc.
   - Examples: fork(), exit(), wait(), etc.

2) File management
   - Create, delete; Open, close; Read, write, reposition; Get and set file attributes, etc.
   - Examples: open(), read(), write(), close(), etc.

3) Device management (memory, tape drives etc.)
   - Request device; Release device; Read, write, reposition, etc.
   - Examples: ioctl(), read(), write(), etc.

4) Information maintenance
   - Get or set time or date; Get or set process attributes, etc.
   - Examples: getpid(), alarm(), sleep(), etc.

5) Communications
   - Create, delete communication connection; Send and receive messages, etc.
   - Examples: pipe(), shmget(), mmap(), etc.

6) Protection
   - Resource access control
   - Examples: chmod(), umask(), chown(), etc.

System programs or utilities

[redacted]

- Provide a convenient environment for program development and execution
  - Most users’ view of the OS is defined by system programs, not the actual system calls.
  - Some of them are simple user interface to system calls.
- System programs can be divided into:
  - File manipulation: Create, delete, copy, rename, print, dump, list.
  - Status information: Date, time, memory, disk space, number of users.
  - File modification: Text editors to create and modify content of files.
  - Programming-language support: Compilers, interpreters, assemblers.
  - Program loading and execution: Absolute, Relocatable, Overlay loaders.
  - Communication: Programs that provide the mechanism for creating virtual connections among processes, users, and different computer systems.

[redacted]

Operating System Design Goals

- Design of the OS is affected by the hardware choice, and system type (batch, timeshared, etc.)
  - User goals – OS should be convenient to use, easy to learn, reliable, safe, and fast.
  - System goals – OS should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient
- Separation of *policy* and *mechanism* is a very important principle
  - It allows maximum flexibility if policy decisions are to be changed later.
  - Mechanisms determine *how* to do something
  - Policies decide *what* will be done.
- Policies are likely to change from place to place and time to time, and therefore a general mechanism would be more desirable.
- Example: A mechanism for ensuring CPU protection is the timer construct; and the decision on how long the timer is set for a particular user is a policy decision.

OS Operation

[redacted]

- In multiprogramming, OS must ensure that an incorrect (or malicious) program cannot cause other programs to execute incorrectly.
- Many programming errors are detected by the hardware, and the errors are normally handled by the OS.

<u>Dual-mode operation</u>

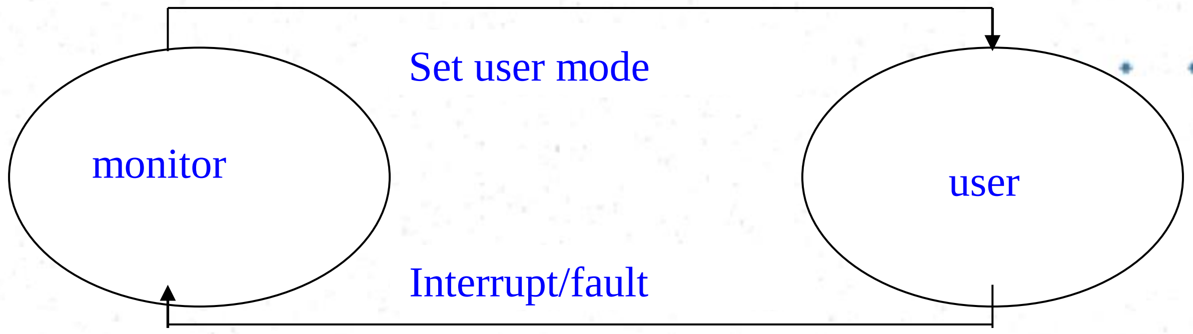
- Hardware provides at least two modes of operations:
  - User mode – in which user programs run.
  - Monitor mode, also called Supervisor, System, or privileged mode.
- Mode bit is provided by the hardware to indicate the current mode: monitor (0) or user (1).

[redacted]

OS Operation (cont.)

Dual-mode operation (cont.)

- Some machine instructions that may cause harm are designated (by hardware) as privileged instructions.
  - E.g., the MSB of the machine code of the instruction is a bit ‘1’
- A privileged instruction can be executed only in monitor mode.
- At system boot-time the hardware starts in monitor mode → OS is loaded.
- OS starts user processes in user mode; a user process could never gain control of the computer in monitor mode.
- When an interrupt or fault occurs hardware switches to monitor mode.



OS Operation (cont.)

[redacted]

I/O

- All I/O instructions are privileged instructions.
- How does the user program perform I/O ? Use system call.
  - Usually takes the form of a trap to a specific location in the interrupt vector.
  - Control passes through the interrupt vector to a service routine in the OS, and the mode bit is set to a monitor mode.
  - The monitor verifies that the parameters are correct and legal, executes the request, and returns control to the instruction following the system call.

[redacted]

OS Operation (cont.)

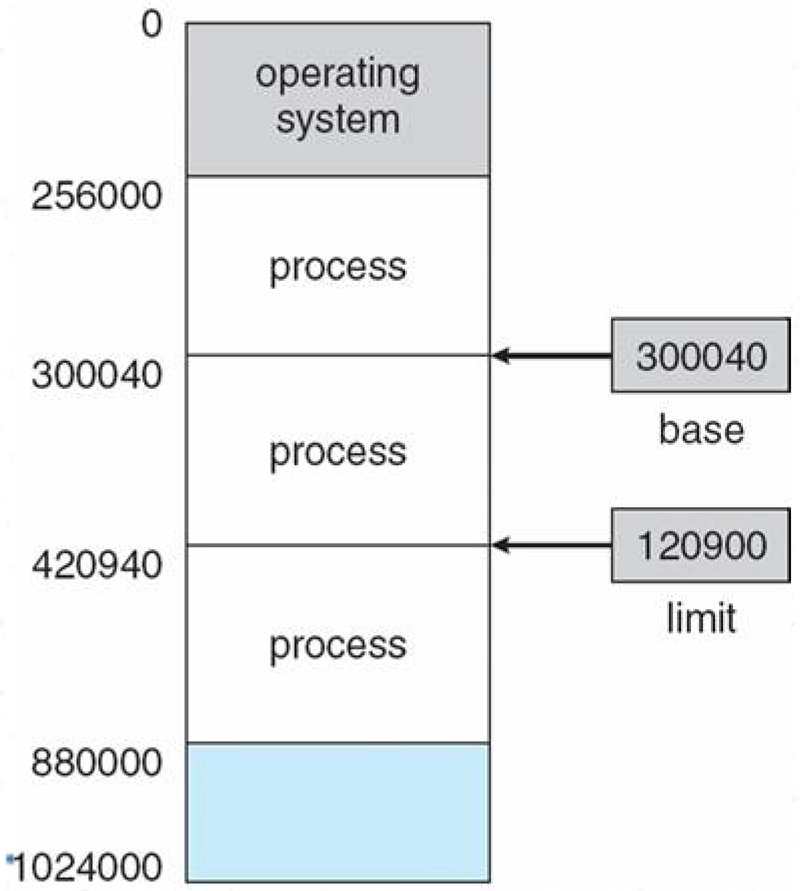
Memory

System must provide memory protection at least for the interrupt vector and the Interrupt Service Routine.

- Use two registers that determine the range of legal addresses a program may access:
  - Base register – holds the smallest legal physical memory address.
  - Limit register – contains the size of the range.
- Memory outside the defined range is protected.
- The load instructions for the base and limit registers are privileged instructions.
- OS has unrestricted access to monitor and user memory.

Memory Protection

[redacted]



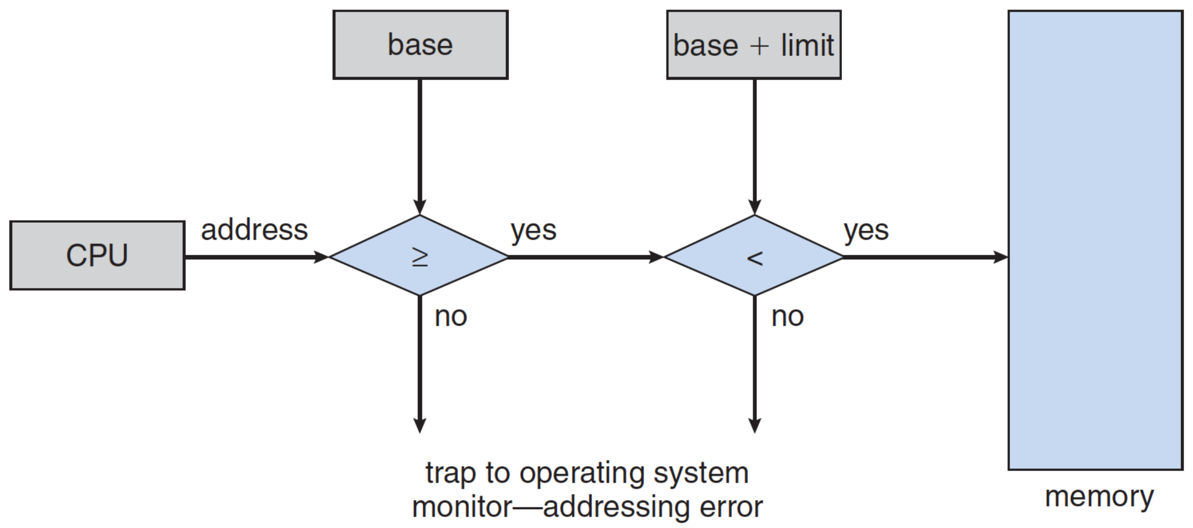


Figure 7.2 Hardware address protection with base and limit registers.

OS Operation (cont.)

[redacted]

CPU

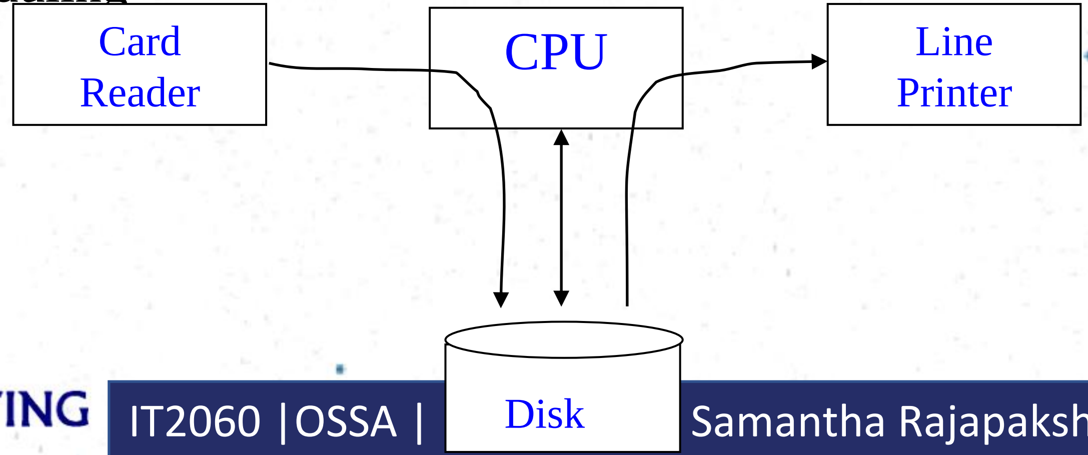
- System must prevent one user program using CPU all the time
  - Because of getting stuck in an infinite loop, or non-fair users
- Use *timer* interrupt after specified period to ensure OS maintains control
  - Timer is decremented every clock tick.
  - When timer reaches value 0, an interrupt occurs.
- Timer is commonly used to implement time sharing.
- Timer is also used to compute the current time.
- Loading timer value is a privileged instruction. Why?

[redacted]

OS development (cont.)

<u>Simultaneous Peripheral Operation OnLine (SPOOL):</u>

- Goal: To keep I/O and CPU busy all the time.
- Use a faster disk as huge buffer → disk was a new technology
- Inputs from cards are read into disks.
- CPU reads input from disks and outputs are written by CPU to disks.
- The I/O of a job is overlapped with its own computation or with another job.
- Need a Job Pool – data structure that allows OS to select job that runs next in order to increase CPU utilization → job scheduling

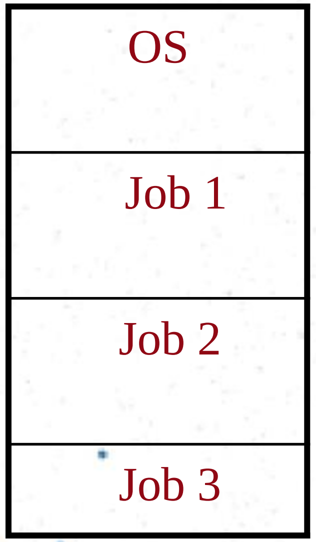


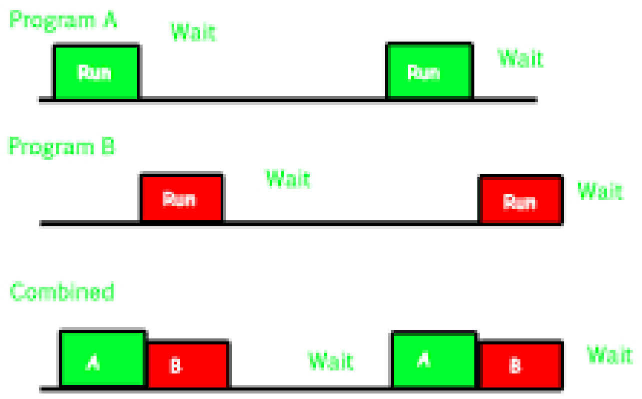
[redacted]

Multiprogramming

- Goal: to increase the CPU utilization
- Several jobs are kept in main memory at the same time, and the CPU is multiplexed among them
- It needs these OS features:
  - Job scheduling → OS chooses jobs in the job pool and put them into memory.
  - Memory management → OS allocates the memory for each job.
  - CPU scheduling → OS chooses one among the jobs in memory (called processes) that is ready to run.
  - I/O allocation.

Memory partition

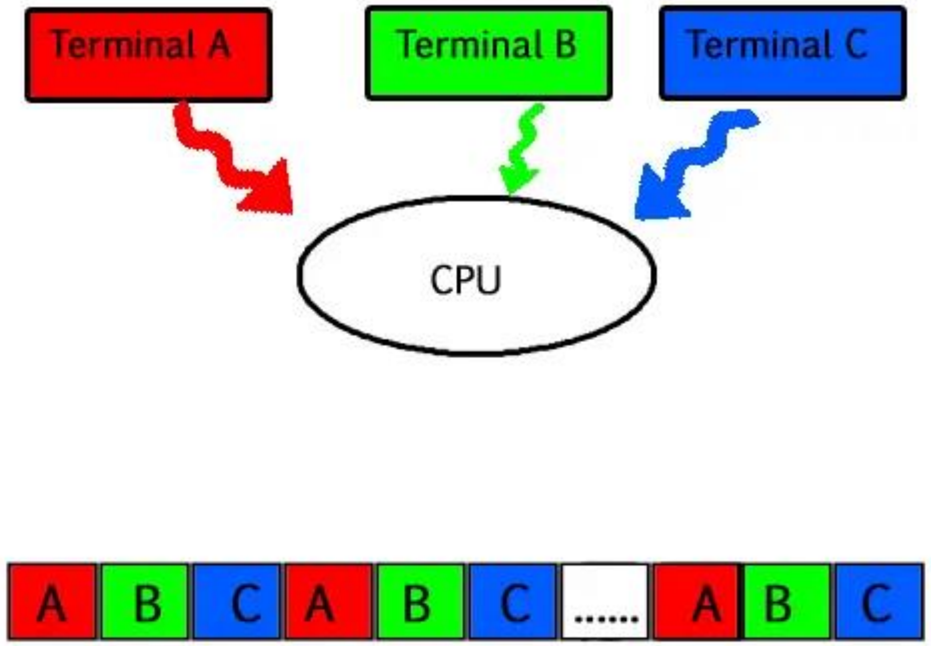




[redacted]

Time-Sharing/Multitasking Systems

- **Goal:** to provide interactive use of computer system at reasonable cost (one computer – several users/jobs).
- It is a variant of multiprogramming → but user input is from on-line terminal.
- CPU is multiplexed among jobs in memory frequently (≤1 sec?) so that users can interact with their running program.
- Today, most systems provide both batch processing and time sharing.



SLIIT FACULTY OF COMPUTING

Real-time Systems

- Well-defined fixed-time constraint – the system is functional if it returns the [redacted] the time constraint.
- Hard real-time system.
  - Guarantees that critical tasks complete on time.
  - Often used as a control device in a dedicated application
    - controlling scientific experiments, medical imaging systems, industrial control systems, and some display systems
  - Secondary storage is limited or absent
    - data is stored in short-term memory, or in ROM.
- Soft real-time system
  - A critical-time task gets priority over others until it completes.
  - Limited utility in industrial control robotics.
  - Useful in applications (multimedia) requiring advanced OS features.

[redacted]

Thank You

SLIIT
FACULTY OF COMPUTING