ICS 212: Operating Systems

Lecture 1

Instructor:

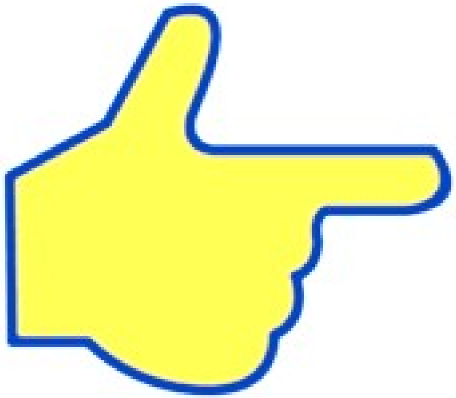
Dr. Nilotpal Chakraborty

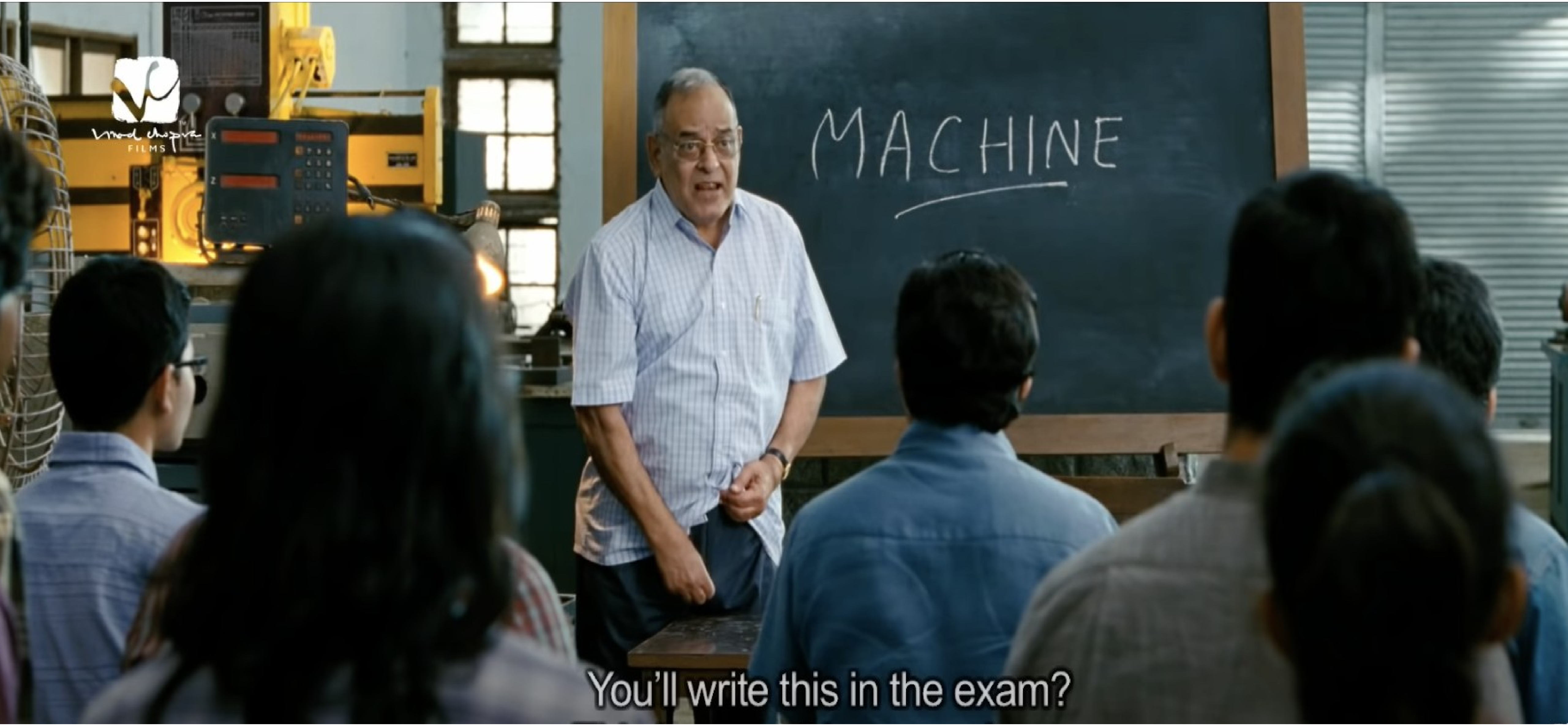
Indian Institute of Information Technology Kottayam

Books and reading materials

- **William Stallings**, *Operating systems: Internals & design principles, Pearson, Seventh edition, 2014.*
- **Silberschatz, Galvin, Gagne**, *Operating System Concepts, Wiley, Ninth Edition, 2016.*
- Other online resources and papers.

Let’s define OS..




MACHINE
You'll write this in the exam?

From the treatise

- Stallings: "An **Operating System** is a program that controls the execution of application programs and acts as an interface between the applications and the computer hardware."

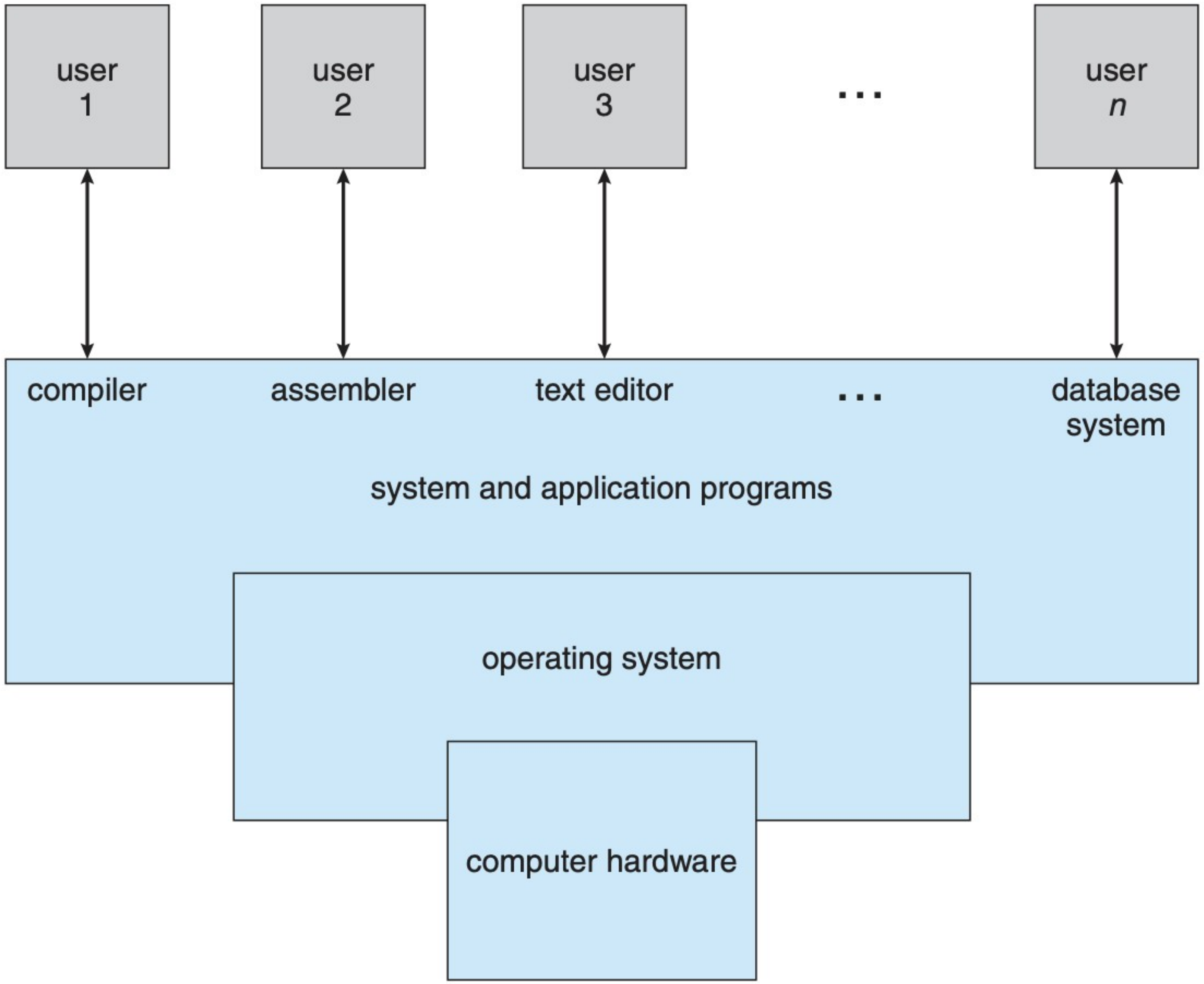
- Galvin: "An **operating system** is a program that manages a computer's hardware. It also provides a basis for application programs and acts as an intermediary between the computer user and the computer hardware."

- **Objectives**: *convenience*, *efficiency*, *evolving*

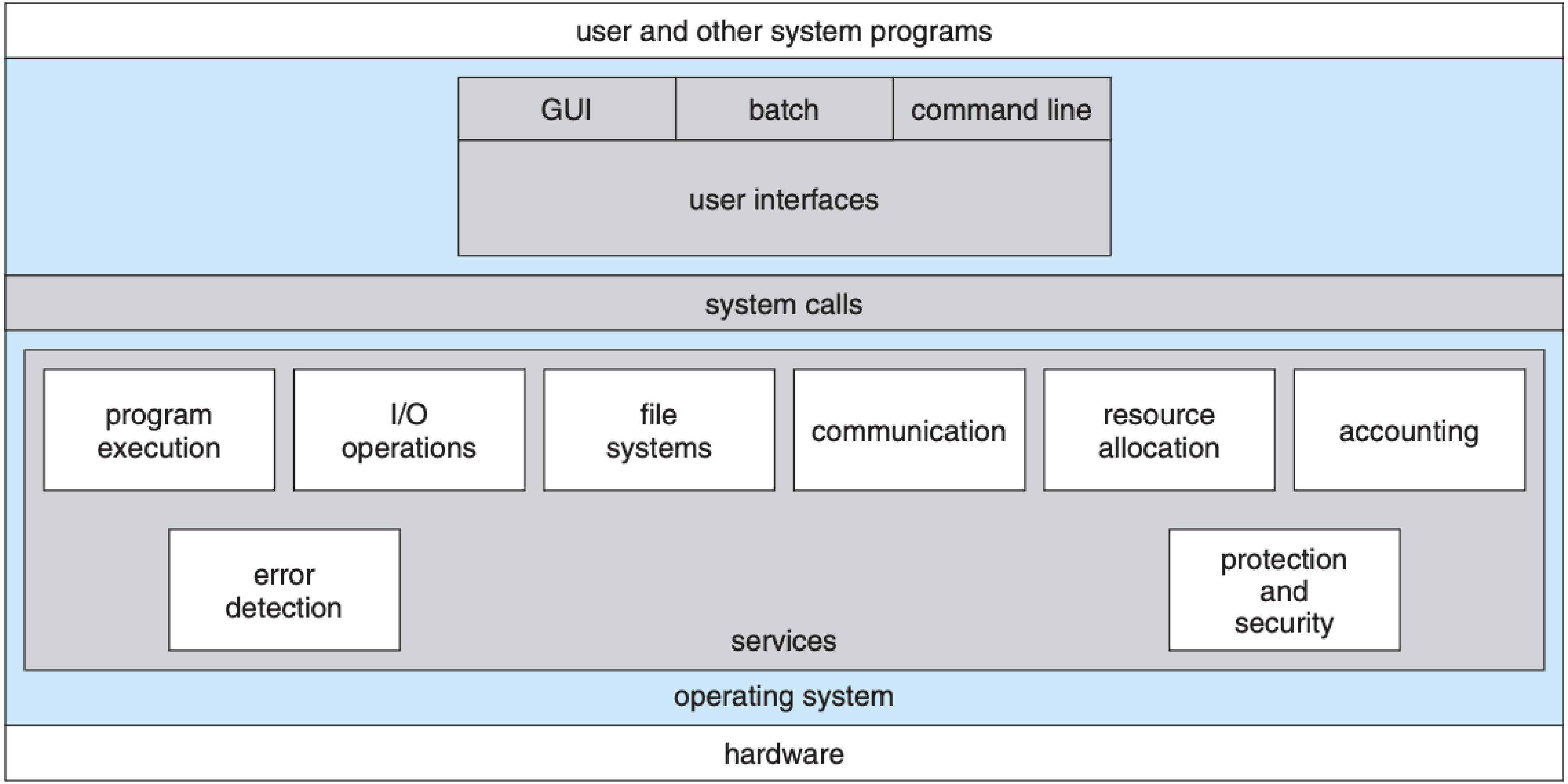
OS as per functionality

- Provides *interface* between hardware and software
- A *resource manager* that allows programs/users to share the hardware resources
- A *controller* that controls execution of programs to prevent errors and improper use of the computer
- Creates a virtual machine that is easier to program than the raw hardware
- Overall services by OS: *program development and execution*, *device control and access*, *error control*, *accounting*, *API*, etc.

Abstract View



OS services



user and other system programs
GUI
batch
command line
user interfaces
system calls
program execution
I/O operations
file systems
communication
resource allocation
accounting
error detection
protection and security
services
operating system
hardware

Over the years..

- Single user computers:
  - One user at a time using the console
  - One function at a time for the computer
- Batch processing:
  - Executes multiple jobs in batches
  - Jobs are submitted on cards or tapes, which are in turn batched sequentially
- Multiprogrammed Batch Systems:
  - Overlap I/O jobs and computing jobs
  - CPU can execute another job while waiting for another job
- Multiprogramming:
  - Several programs can run simultaneously
  - OS manages interactions

Thank You!

ICS 212: Operating Systems

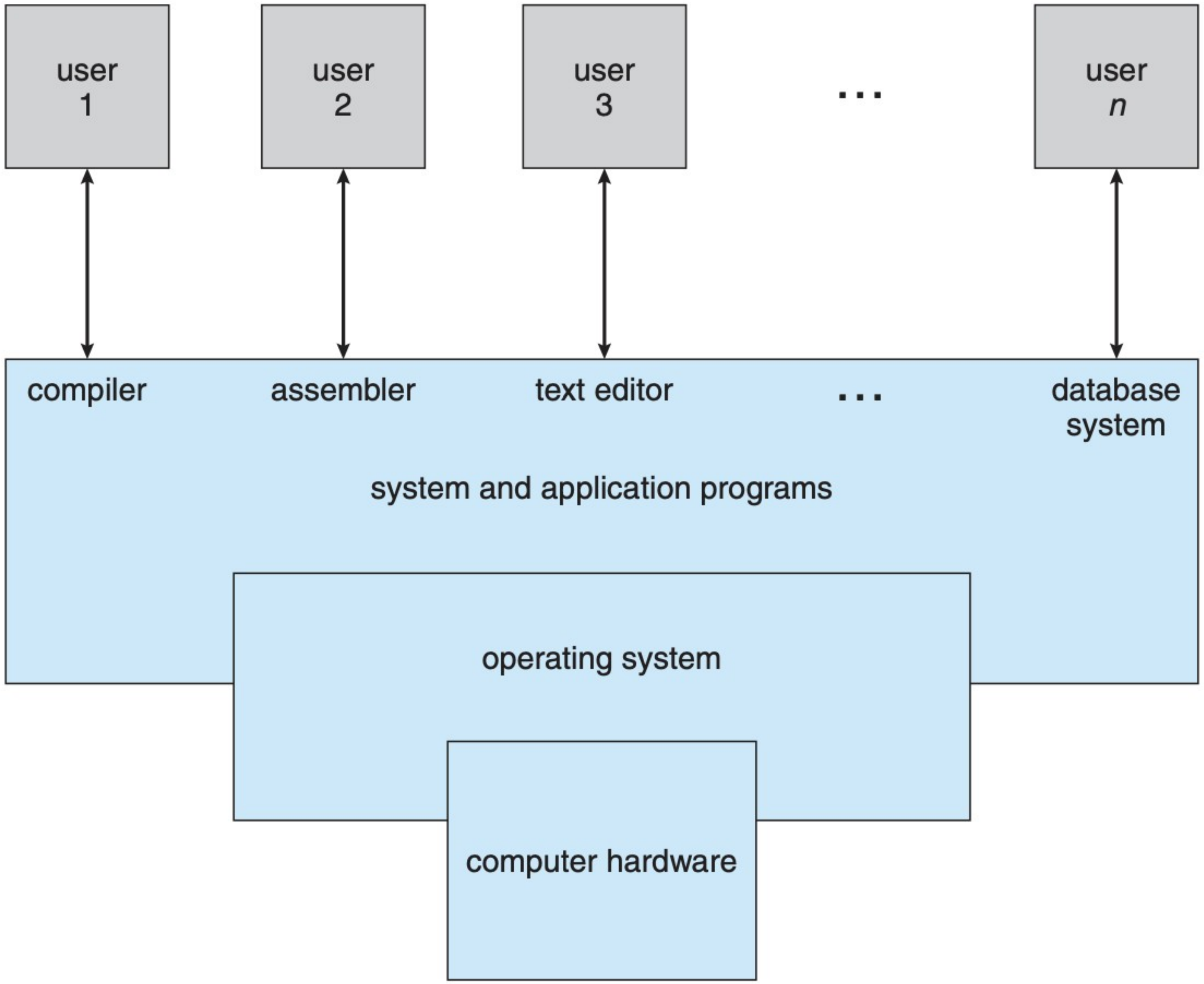
Lecture 2

Instructor:

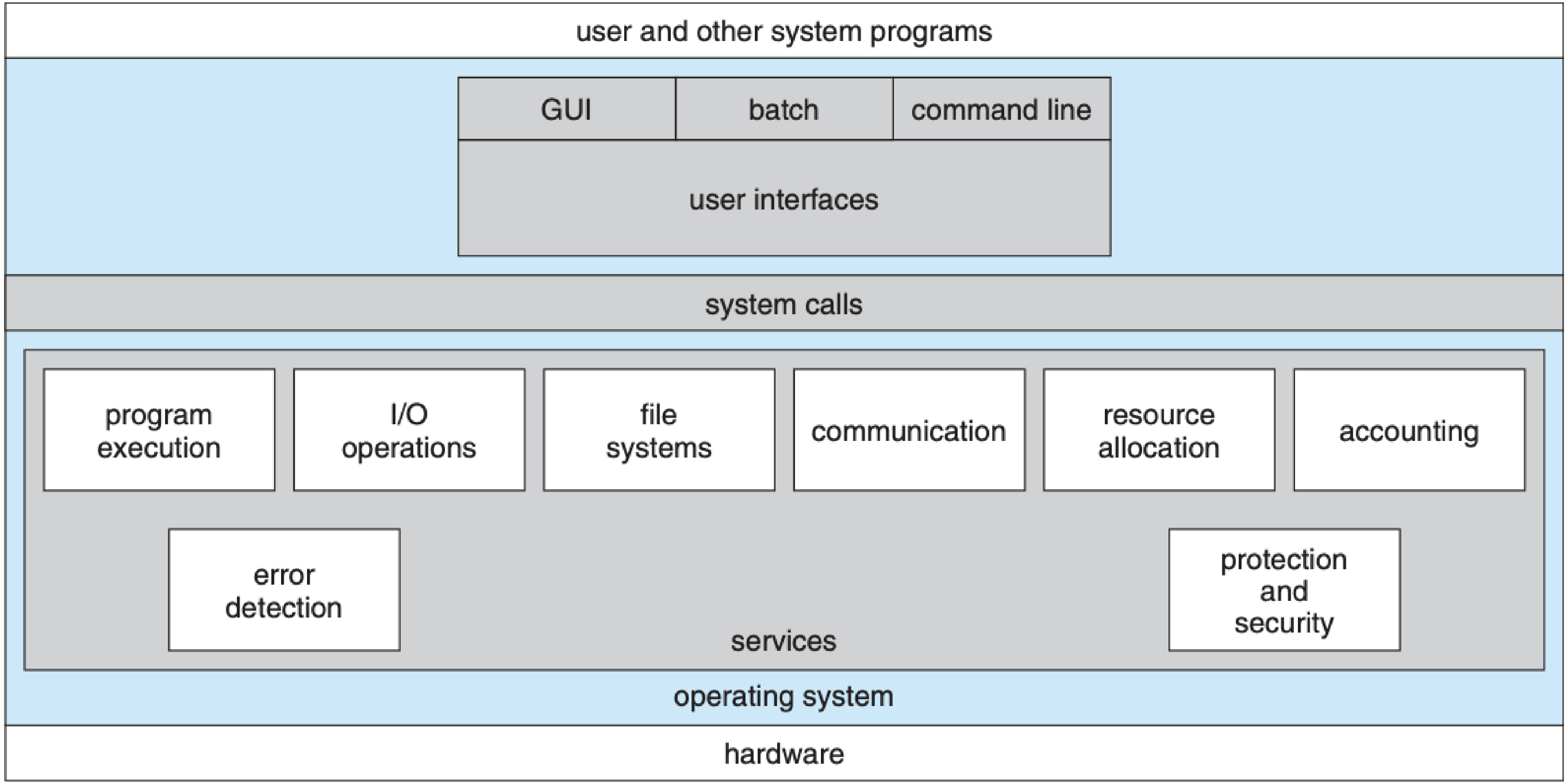
Dr. Nilotpal Chakraborty

Indian Institute of Information Technology Kottayam

Abstract View



OS services


user and other system programs
GUI
batch
command line
user interfaces
system calls
program execution
I/O operations
file systems
communication
resource allocation
accounting
error detection
protection and security
services
operating system
hardware

OS Timeline

Timeline of Operating Systems

1956 – GM-NAA I/O, produced by General Motors for its IBM 704

1977 – Apple][released

1981 – MS-DOS is released by Microsoft

1991 – Linux is released by Linus Torvalds

1995 – Windows 95 is released

2008 – Android is released (based on a Linux kernel)

2011 – OpenShift released by Red Hat

1960s – IBM develops a series of OSs for its 360 series. Multics is developed and abandoned but UNIX is developed as a consequence.

1970s – Unix becomes popular in academic circles and spawns many versions

1980s – The home computer revolution

1990s – Windows dominates the laptop and desktop market

2000s – Unix and then Linux dominate the Supercomputer Market

2010s – Smart phones become ubiquitous after the iPhone release in 2007

Computer System Organization

Architectural features

OS service	Hardware support
Interrupts	Interrupt vectors
Protection	Kernel/user mode, protected instructions, base/limit registers
System calls	Trap instructions, Trap vectors
I/O	Interrupts and memory mapping
Virtual memory	Translation lookaside buffers

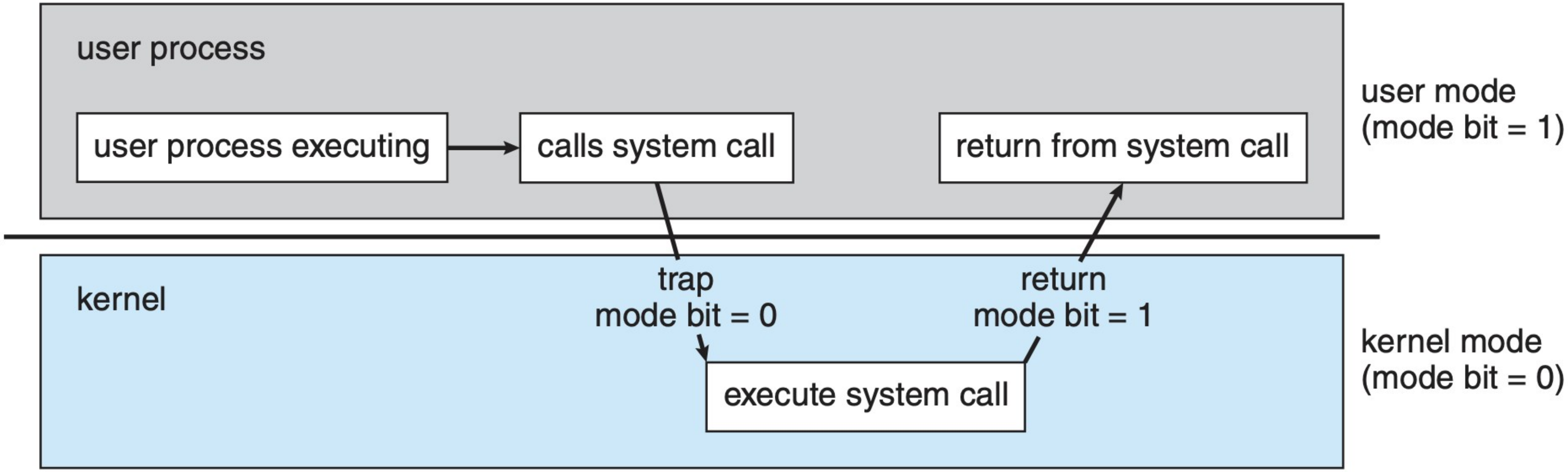
Functionalities of Interrupts

- **An operating system is interrupt driven**
- **Interrupt transfers control to the interrupt service routine generally, through the interrupt vector, which contains the addresses of all the service routines**
- **Interrupt architecture must save the address of the interrupted instruction**
- **Incoming interrupts are disabled while another interrupt is being processed to prevent a lost interrupt**

Protection

Crossing the boundaries

- **System calls: OS procedures that execute privileged instructions**



- **A *trap* is a software-generated interrupt caused either by an error or a user request**

System calls for UNIX

System Call	Description
access()	This checks if a calling process has access to the required file
chdir()	The chdir command changes the current directory of the system
chmod()	The mode of a file can be changed using this command
chown()	This changes the ownership of a particular file
kill()	This system call sends kill signal to one or more processes
link()	A new file name is linked to an existing file using link system call.
open()	This opens a file for the reading or writing process
pause()	The pause call suspends a file until a particular signal occurs.
stime()	This system call sets the correct time.
times()	Gets the parent and child process times
alarm()	The alarm system call sets the alarm clock of a process
fork()	A new process is created using this command
chroot()	This changes the root directory of a file.
exit()	The exit system call is used to exit a process.

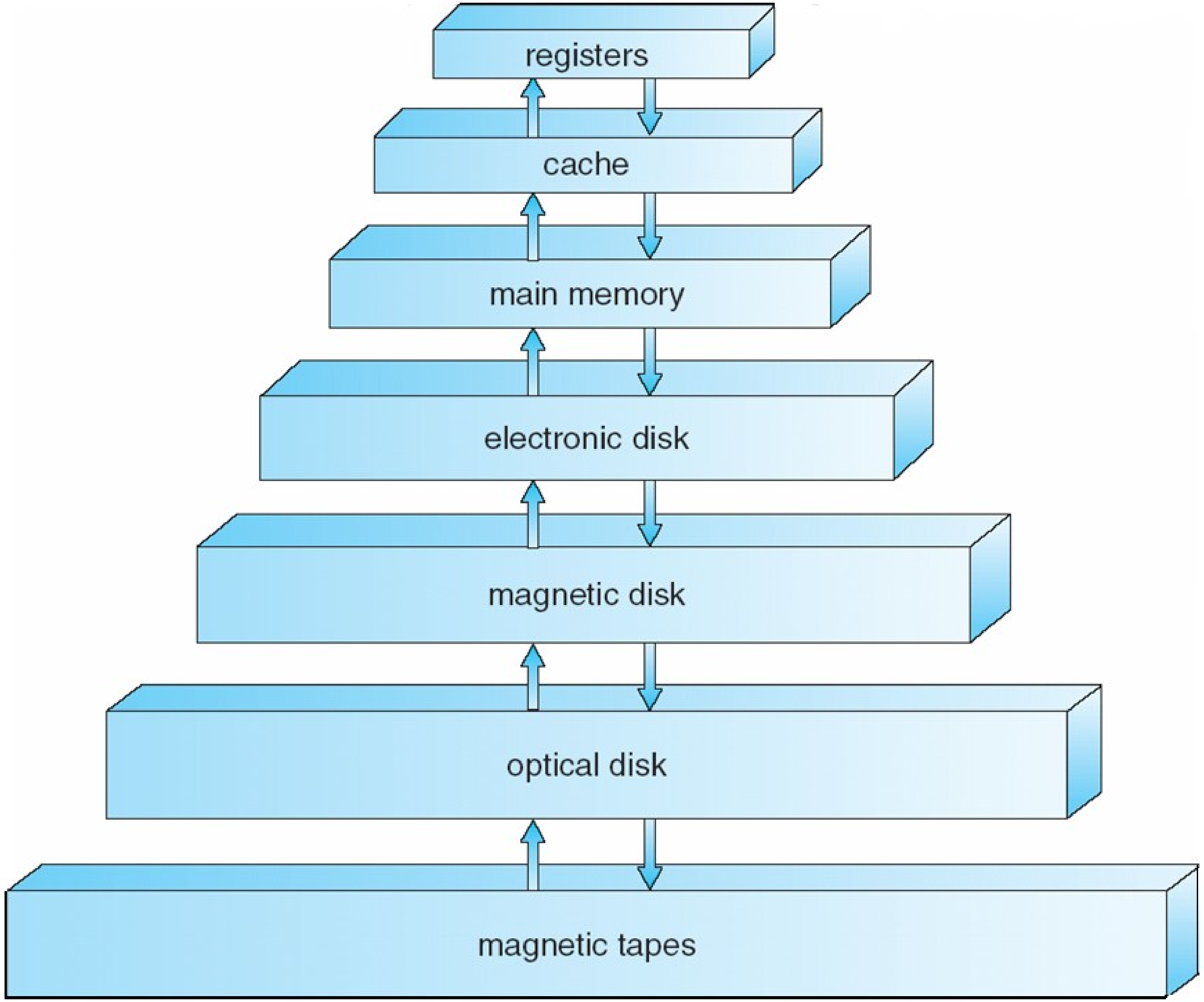
Memory protection

- Architecture must provide support for the OS to:
  - Protect user programs from each other
  - Protect OS from the user programs
- Solution: Based register, Limit register

Memory Structure

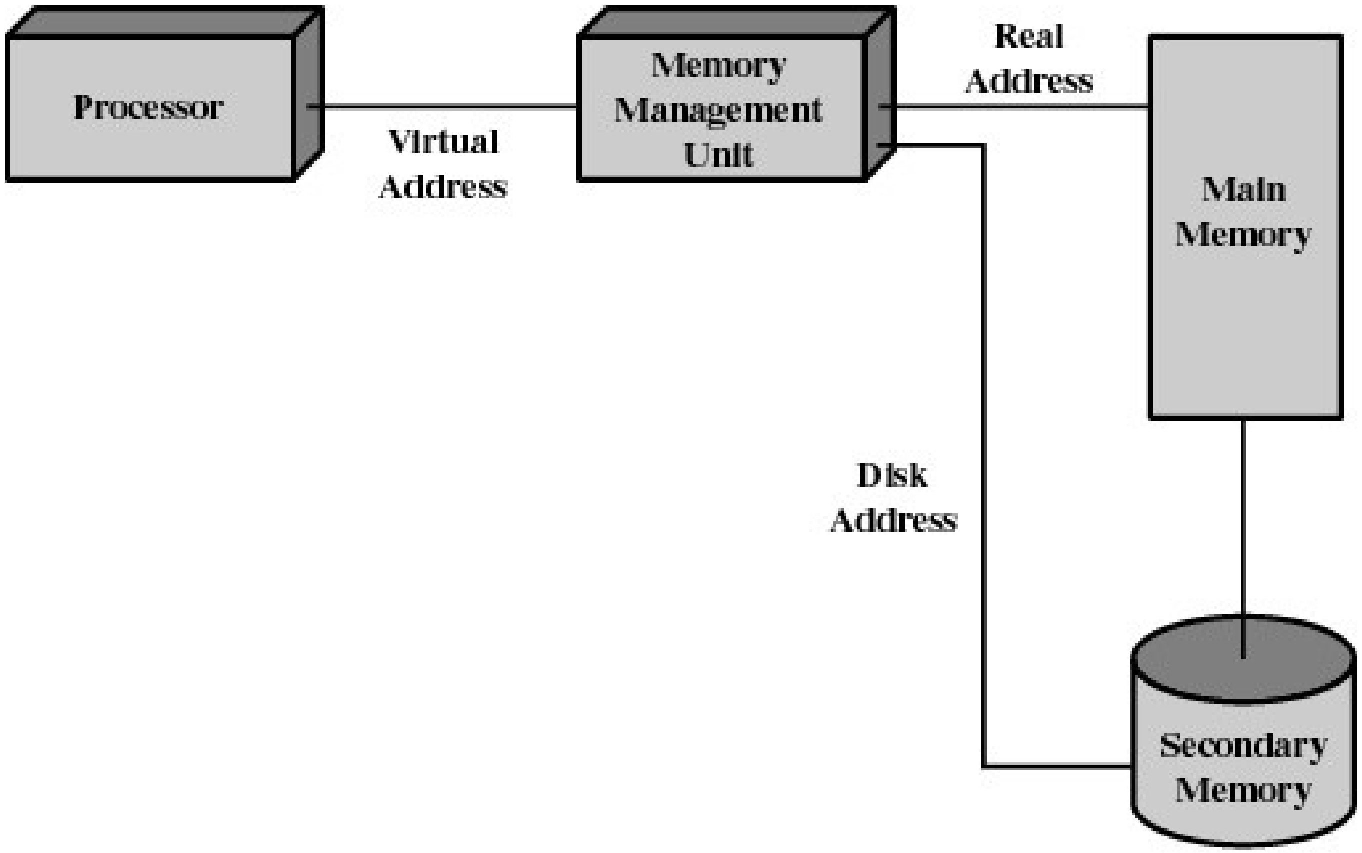
- **Main memory – only large storage media that the CPU can access directly**
- **Secondary storage – extension of main memory that provides large nonvolatile storage capacity**
- **Magnetic disks – rigid metal or glass platters covered with magnetic recording material**
  - **Disk surface is logically divided into tracks, which are subdivided into sectors**
  - **The disk controller determines the logical interaction between the device and the computer**

Storage-Device Hierarchy



Virtual memory

- Allows users to ***run programs without loading it completely***.
- OS ***keeps track of the individual parts*** of the programs
- **Translation lookaside buffers**



ICS 212: Operating Systems

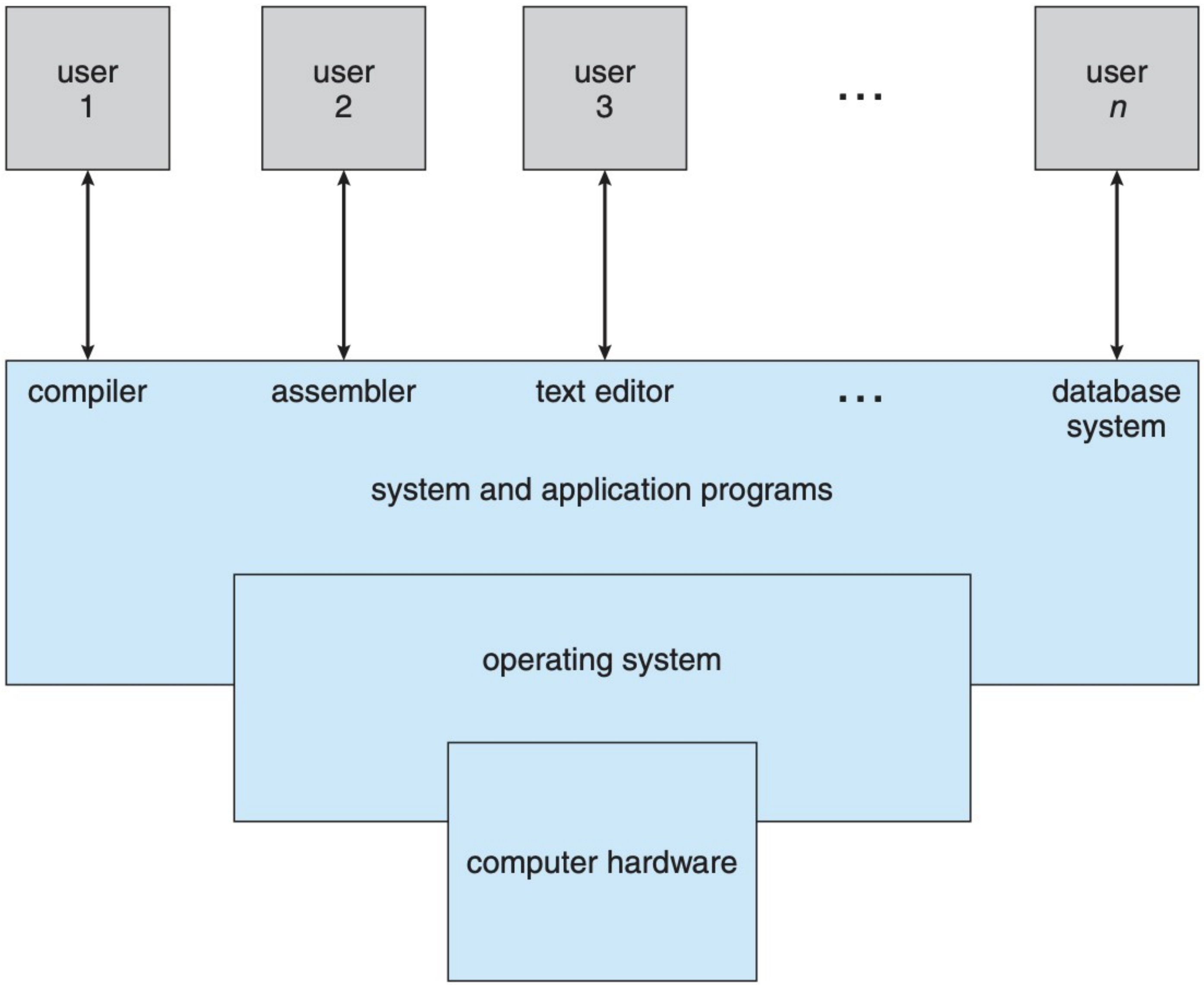
Lecture 3

Instructor:

Dr. Nilotpal Chakraborty

Indian Institute of Information Technology Kottayam

Abstract View of OS



Computer System

Registers

- **Dedicated name for one word of memory in the CPU**
- ***General purpose*** **registers**
- ***Special purpose*** **registers**

I/O Operation

- **I/O devices and the CPU can execute concurrently**
- **Each *device controller* is in-charge of a particular device type**
- **Each device controller has a local buffer**
- **CPU moves data from/to main memory to/from local buffers**
- **I/O is from the device to local buffer of controller**
- **I/O techniques: *programmed I/O*, *Interrupt-driven I/O*, *direct memory access (DMA)***

Main OS Concepts

- ***Process*** **Management**
- ***Memory*** **Management**
- **Information protection and security**
- ***Scheduling*** **and resource management**
- **System structure**

Process Management

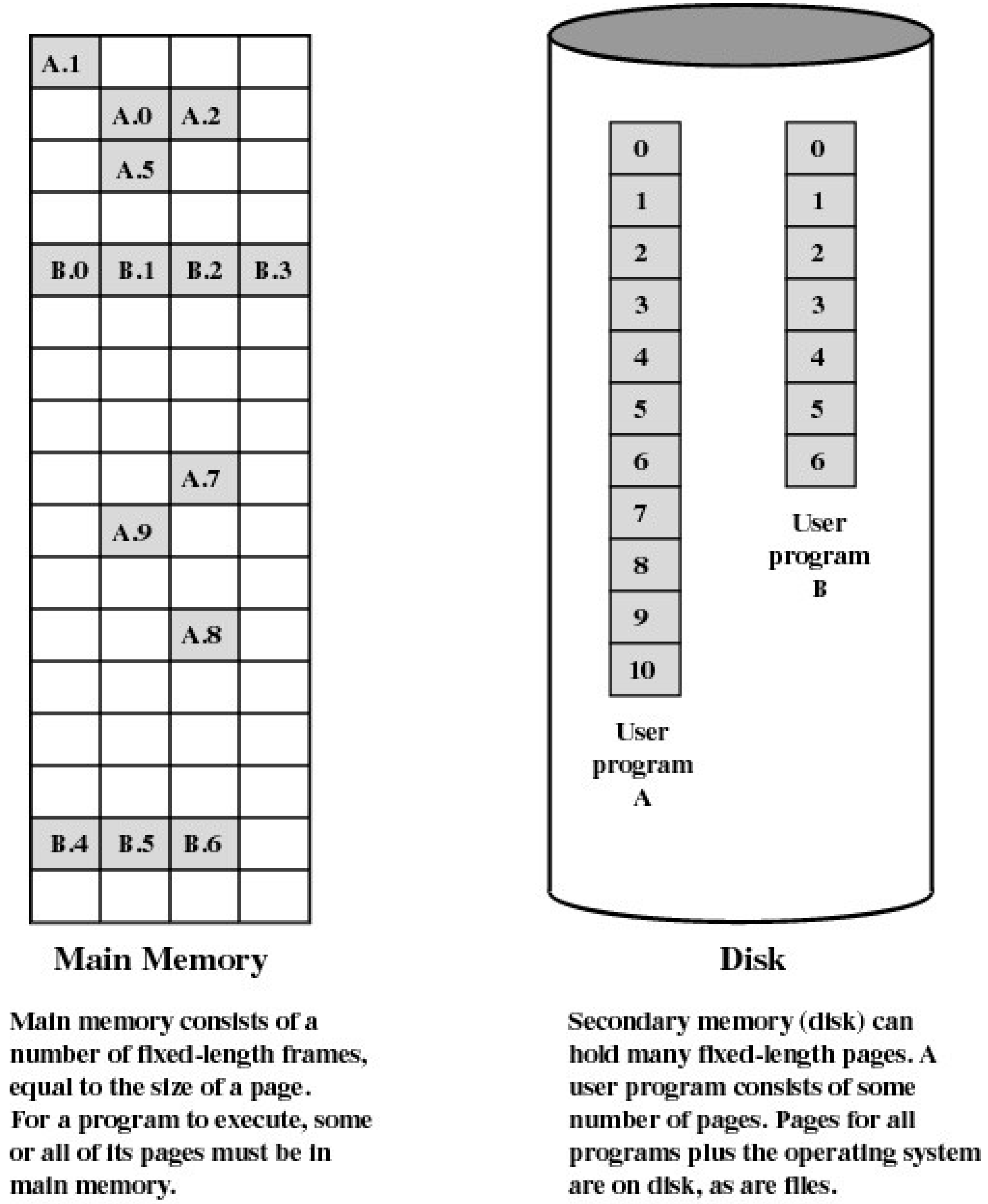
- **Process: a *program in execution***
- **An instance of a program running on a computer**
- **The entity that can be assigned to and executed on a processor**
- **A unit of activity characterized by a single sequential thread of execution, a current state, and an associated set of system resources**

Memory Management

- **Process isolation**
- **Automatic allocation and management**
- **Support for modular programming**
- **Protection and access control**
- **Long-term storage**

Paging

- Allows process to be comprised of a number of fixed-size blocks, called pages
- *Virtual address* is a page number and an offset within the page
- Each page may be located any where in main memory
- Real address or physical address in



Information Protection and Security

- *Access control*
  - Regulate user access to the system
- *Information flow control*
  - Regulate flow of data within the system and its delivery to users
- *Certification*
  - Proving that access and flow control perform according to specifications

Scheduling and Resource Management

- ***Fairness***
  - **Give equal and fair access to all processes**
- ***Differential responsiveness***
  - **Discriminate between different classes of jobs/users**
- ***Efficiency***
  - **Maximize throughput, minimize response time, and accommodate as many uses as possible**

ICS 212: Operating Systems

Lecture 4

Instructor:

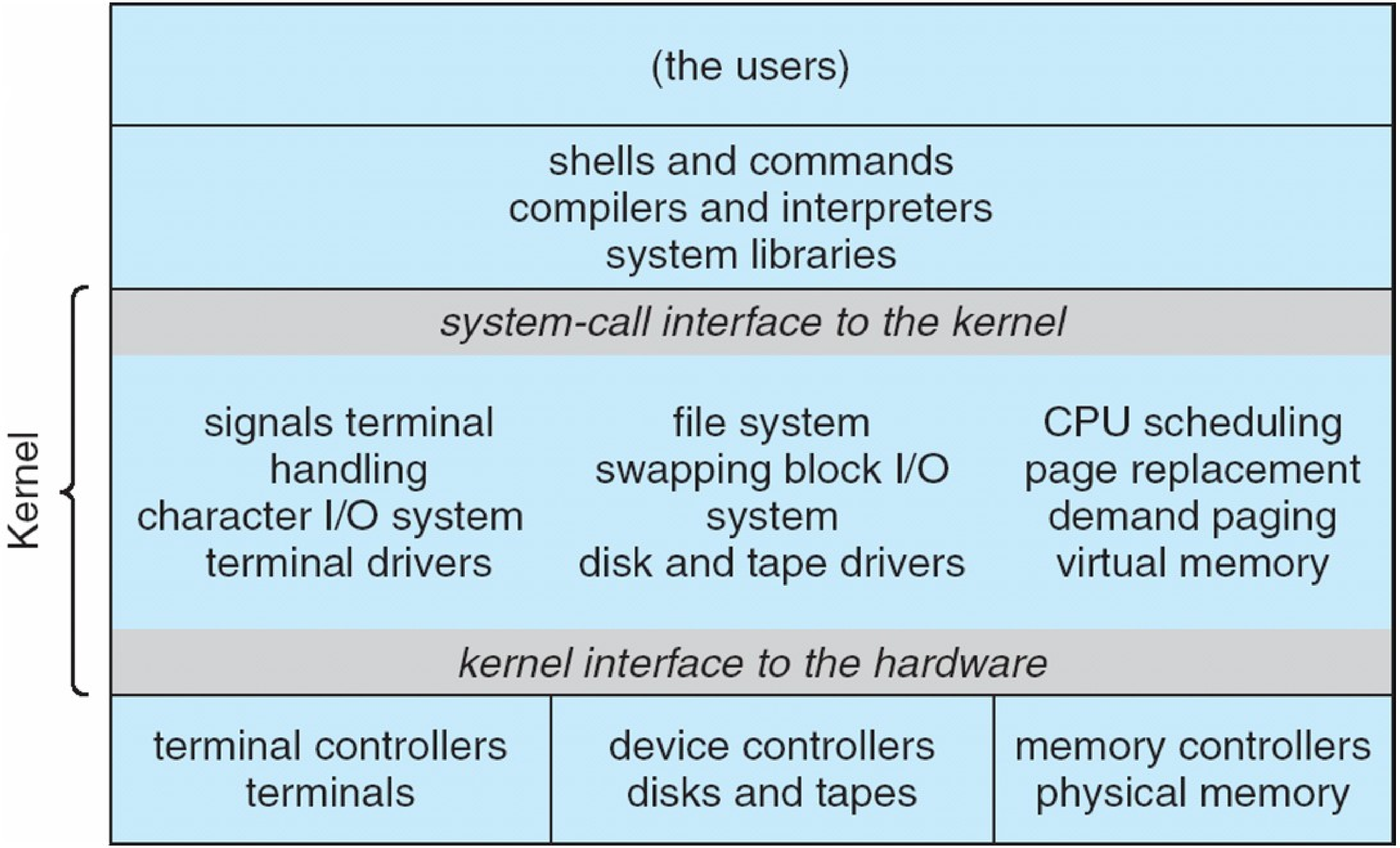
Dr. Nilotpal Chakraborty

Indian Institute of Information Technology Kottayam

OS Structure

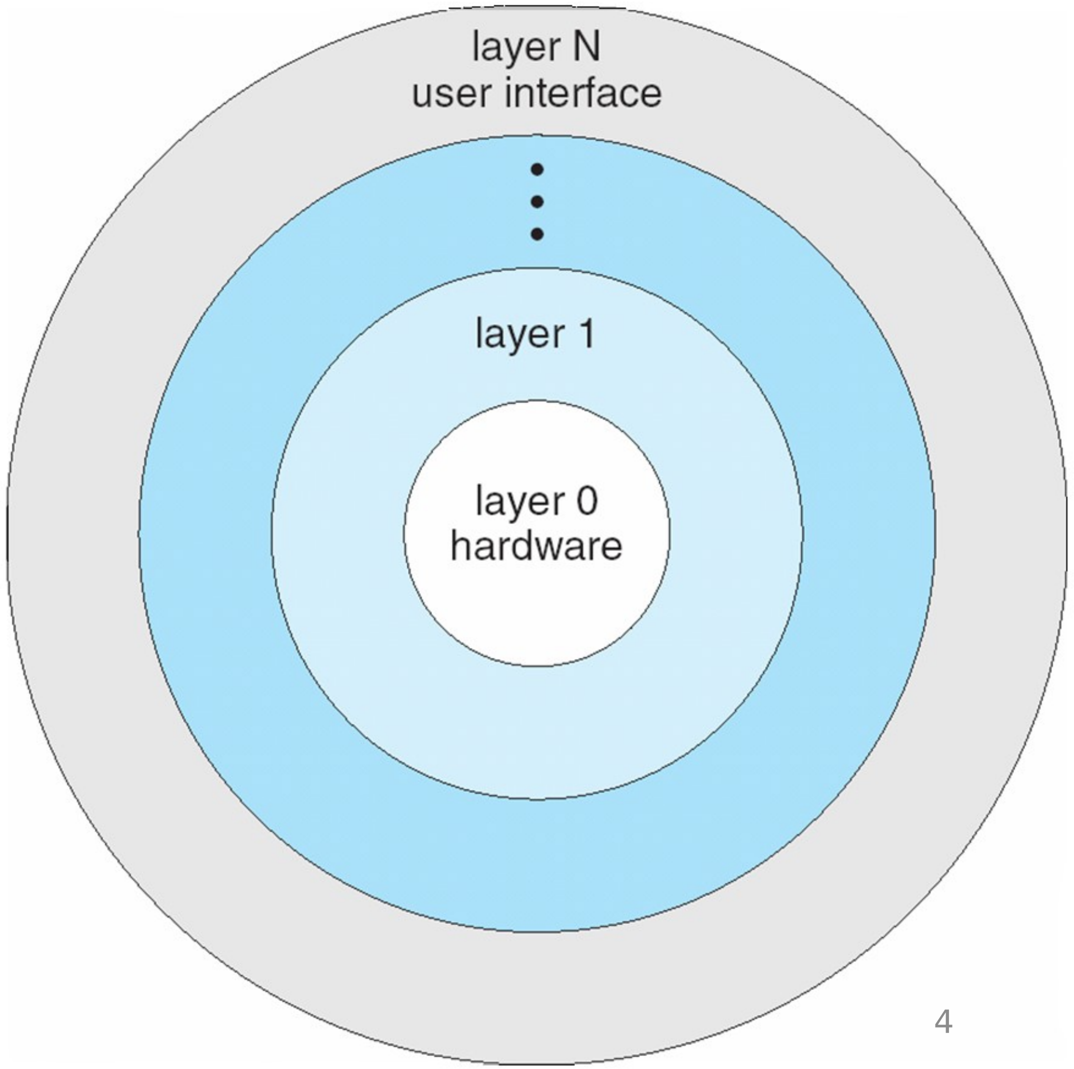
- **Kernel mode: user programs access via *system calls***
- **Architectures:**
  - ***Monolithic kernel***
  - ***Layered architecture***
  - ***Microkernel***
  - ***Modular approach***

Monolithic kernel



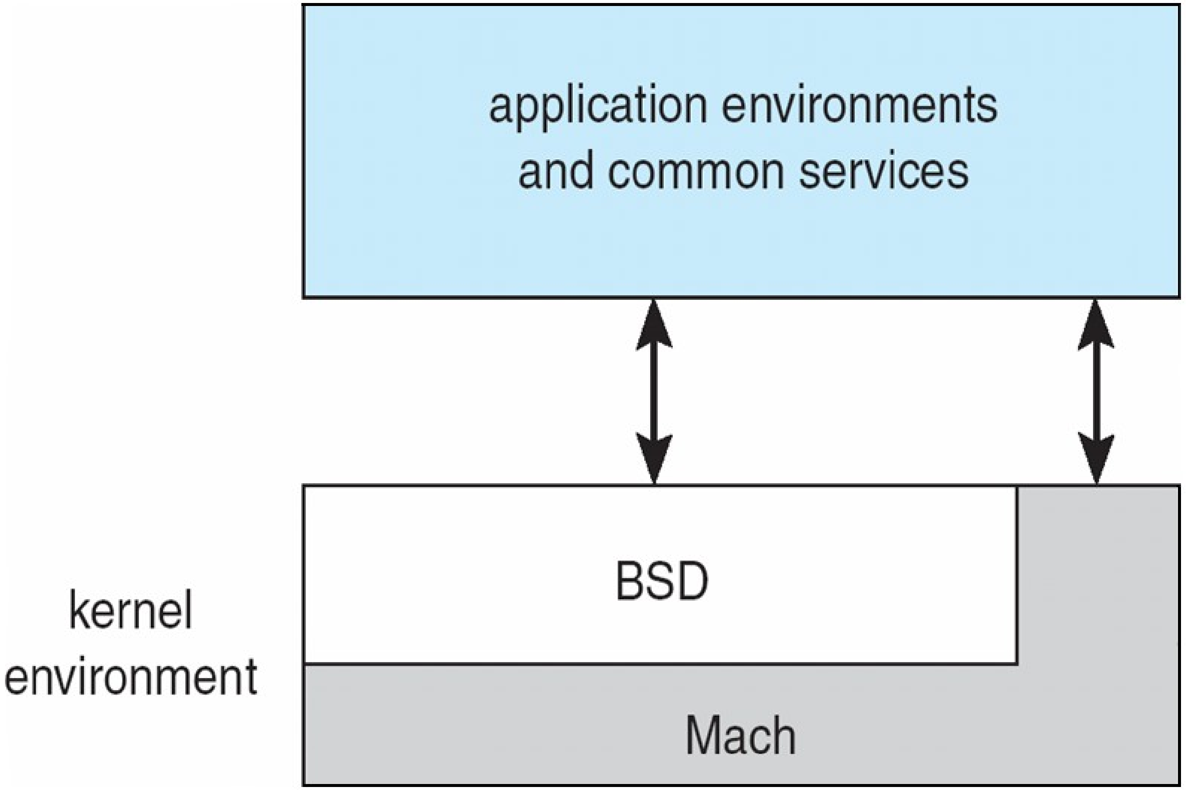
Layered architecture

- **Leyer uses Layer and provides services to Layer**



Microkernel

- **Moves as much from the kernel into “*user*” space**



Benefits:

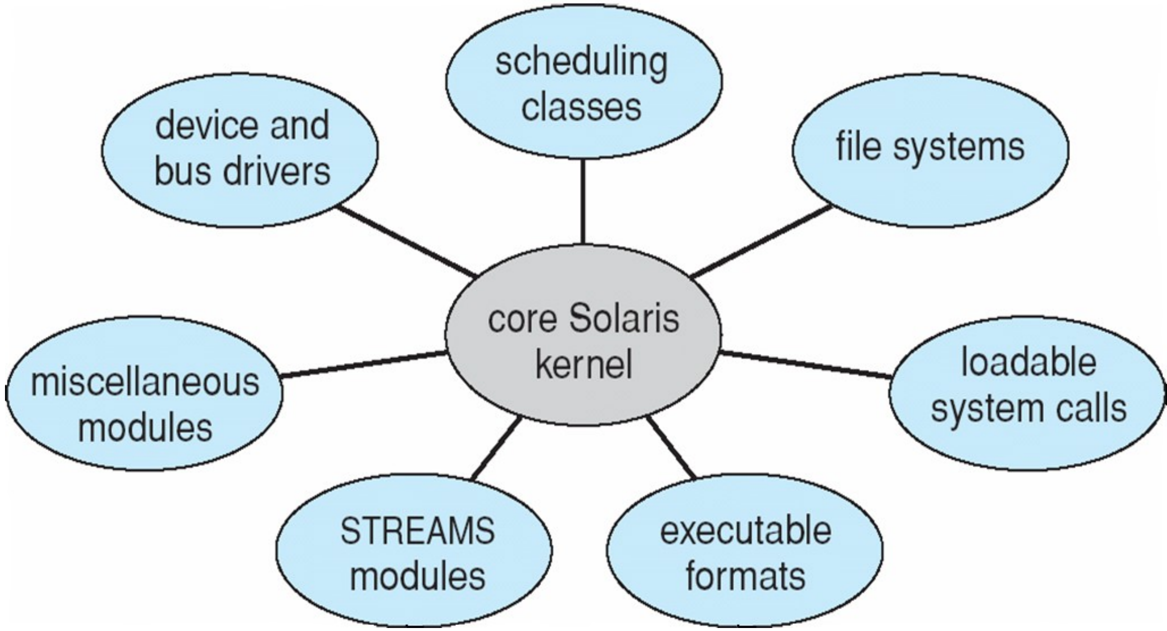
- **Easier to extend a microkernel**
- **Easier to port the operating system to new architectures**
- **More reliable (less code is running in kernel mode)**
- **More secure**

Disadvantage:

- **Performance overhead of user space to kernel space communication**

Modular approach

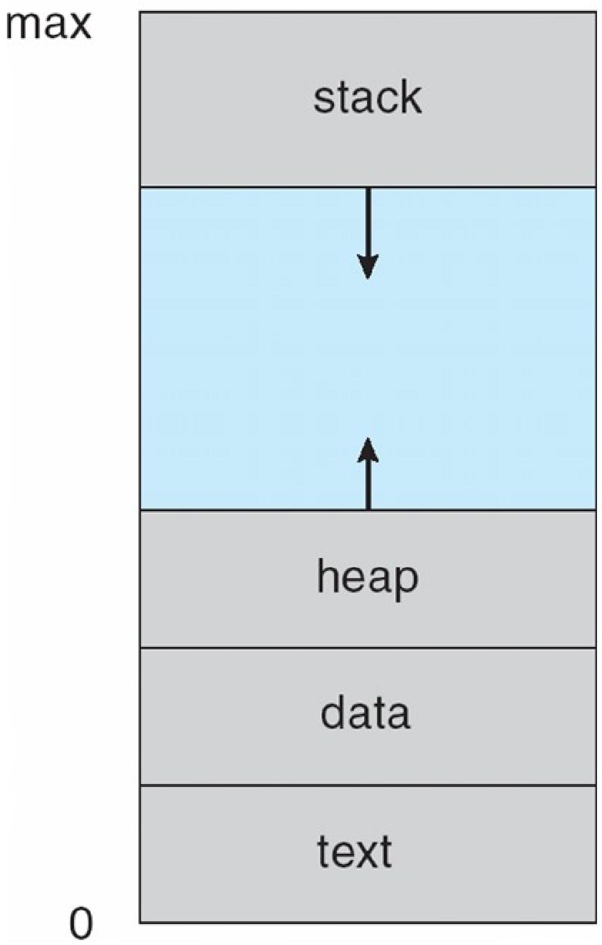
- **Most modern operating systems implement kernel modules**
  - **Uses object-oriented approach**
  - **Each core component is separate**
  - **Each talks to the others over known interfaces**
  - **Each is loadable as needed within the kernel**
- **Overall, similar to layers but with more flexible**



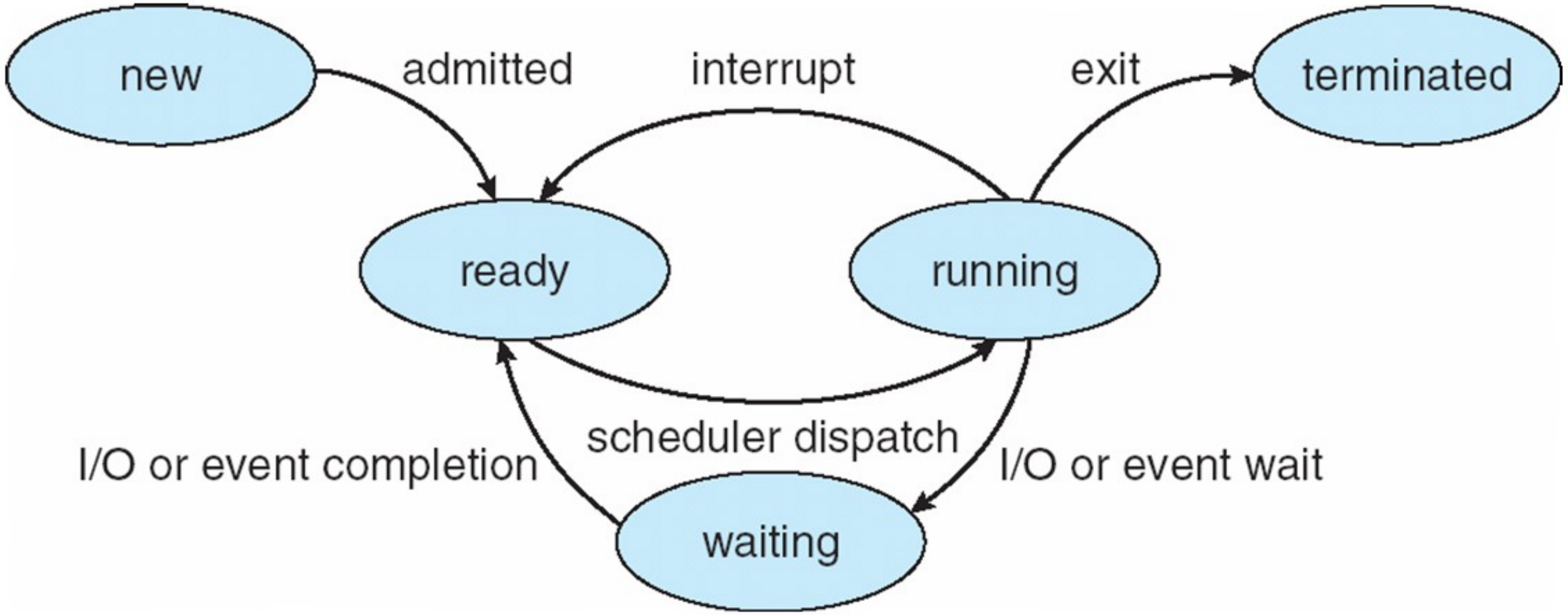
Processes

- **An operating system executes a variety of programs:**
  - **Batch system – jobs**
  - **Time-shared systems – user programs or tasks**
- **Process – *a program in execution***
- **Also called a *task***
- **Can be traced**
  - **List the sequence of instructions that execute**
- **A process includes:**
  - ***Program counter***
  - ***Stack pointer***
  - ***Data section***
  - ***etc.. (more on this soon)***

Process in Memory



Process state

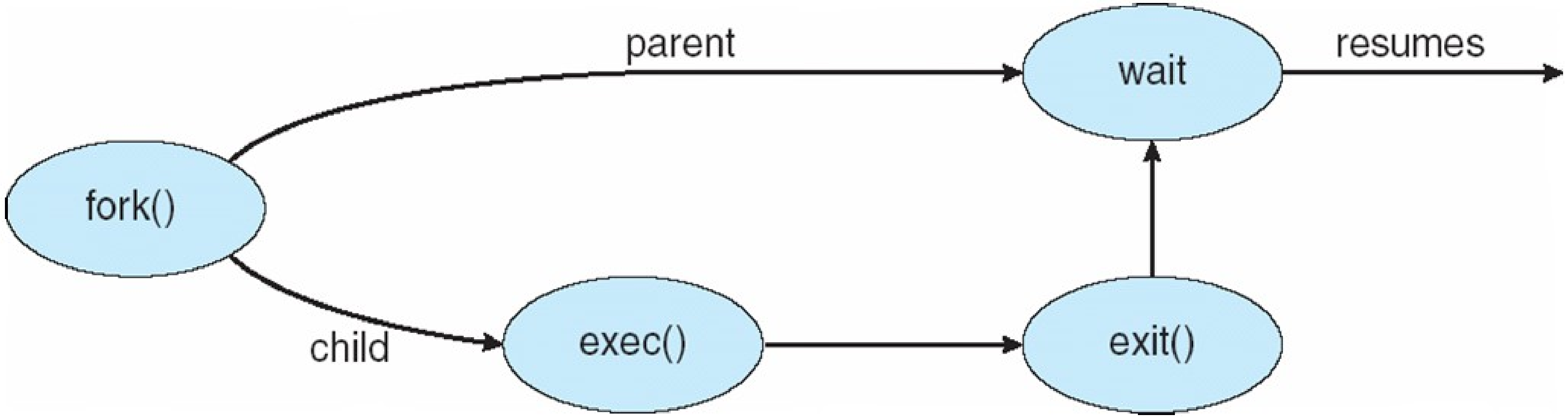


Process Creation

- ***Parent process create children processes***, **which, in turn create other processes, forming a tree of processes**
- **Generally, process identified and managed via a *process identifier (pid)***
- **Resource sharing**
  - **Parent and children share all resources**
  - **Children share subset of parent's resources**
  - **Parent and child share no resources**
- **Execution**
  - **Parent and children execute concurrently**
  - **Parent waits until children terminate**

Process Creation

- **Address space**
  - ***Child duplicate of parent***
  - **Child has a program loaded into it**
- **UNIX examples**
  - **fork system call creates new process**
  - **exec system call used after a fork to replace the process’ memory space with a new program**



Process Termination

- Process executes last statement and asks the operating system to delete it (exit)
  - Output data from child to parent (via wait)
  - Process’ resources are deallocated by operating system
- Parent may terminate execution of children processes (abort)
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - *If parent is exiting*
    - Some operating system do not allow child to continue if its parent terminates
      - All children terminated - cascading termination

ICS 212: Operating Systems

Lecture 5

Instructor:

Dr. Nilotpal Chakraborty

Indian Institute of Information Technology Kottayam

Last class

- **Process: a *program in execution***
- Process States: ***new, ready, executing, waiting, terminated***
- A process includes:
    - *program counter*
    - *stack*
    - *data section*

Process Control Block

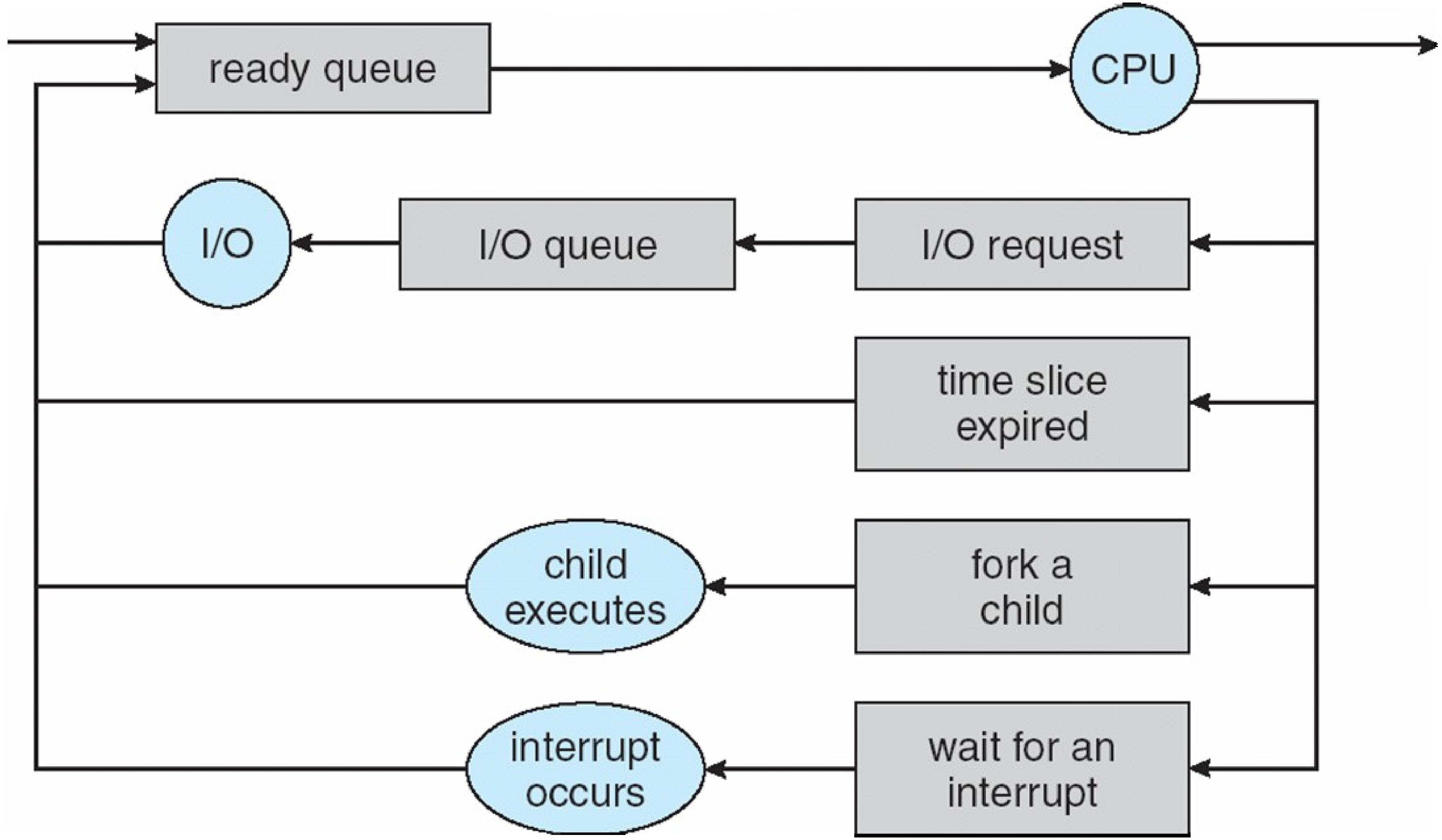
Information associated with each process

- *Process Identifier*
- *Process state*
- *Priority*
- *Program counter*
- *Memory pointers*
- *CPU scheduling information*
- *Accounting information*
- *I/O status information*

Process Scheduling Queues

- ***Job queue*** – set of all processes in the system
- ***Ready queue*** – set of all processes residing in main memory, ready and waiting to execute
- ***Device queues*** – set of processes waiting for an I/O device
- *Processes migrate among the various queues*

Representing Process Scheduling

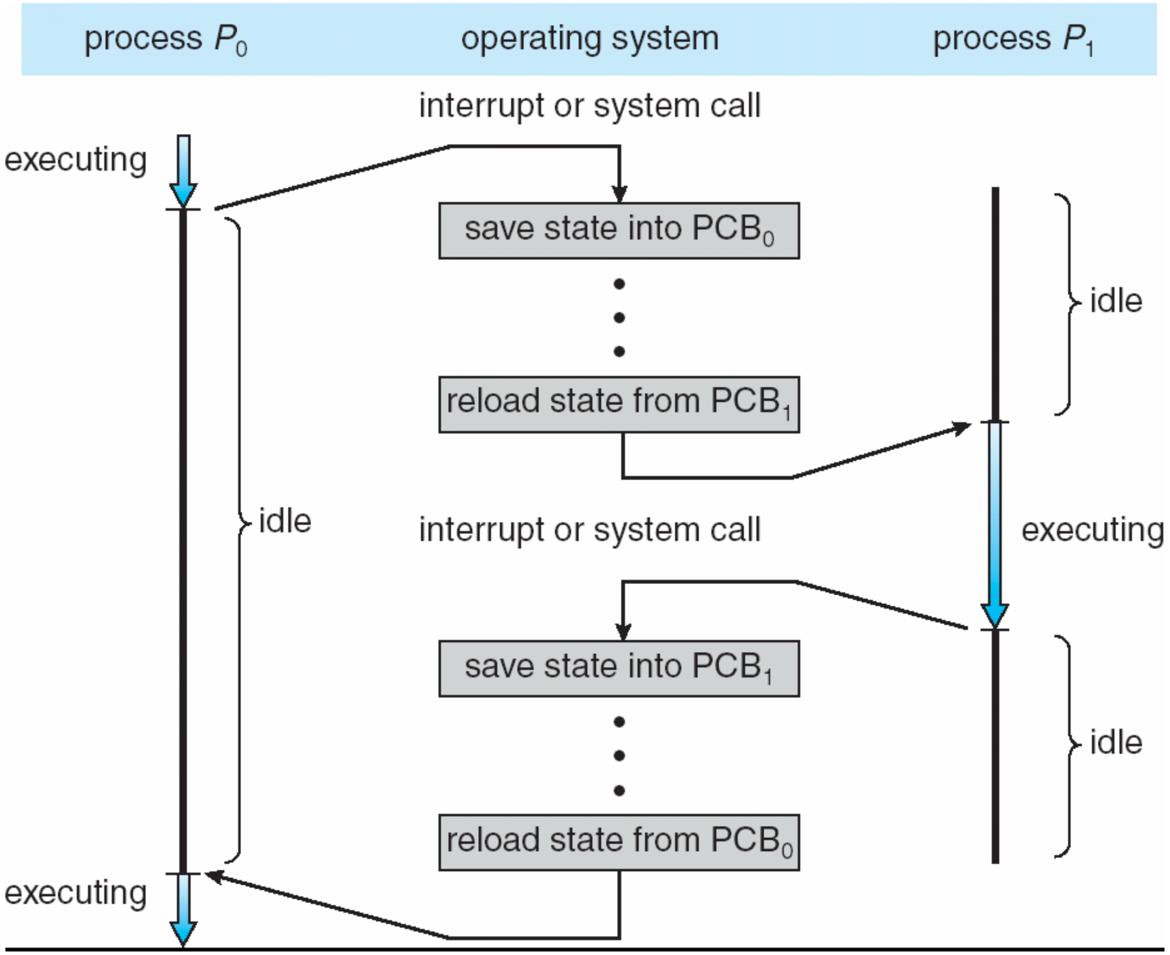


Process Schedulers

- ***Long-term scheduler*** (or job scheduler) – selects which processes should be brought into the ready queue
- ***Short-term scheduler*** (or CPU scheduler) – selects which process should be executed next and allocates CPU
- Processes can be described as either:
    - ***I/O-bound process*** – spends more time doing I/O than computations, many short CPU bursts
    - ***CPU-bound process*** – spends more time doing computations; few very long CPU bursts

Context Switching

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a context switch
- Context of a process represented in the PCB
- Context-switch time is overhead; the system does no useful work while switching
- Time dependent on hardware support


process P_0
operating system
process P_1
interrupt or system call
executing
save state into PCB_0
idle
reload state from PCB_1
idle
interrupt or system call
executing
save state into PCB_1
idle
reload state from PCB_0
executing

Independent and Cooperating Processes

- Processes within a system may be ***independent*** or ***cooperating***
- Independent processes don’t affect each other and don’t share.
- Cooperating process can affect or be affected by other processes, including sharing data
- Reasons for cooperation:
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience
- Cooperating processes need **interprocess communication** (**IPC**)
  - Message Passing
  - Shared memory

ICS 212: Operating Systems

Lecture 6

Instructor:

Dr. Nilotpal Chakraborty

Indian Institute of Information Technology Kottayam

Independent and Cooperating Processes

- **Processes within a system may be *independent* or *cooperating***
- **Independent processes don’t affect each other and don’t share.**
- **Cooperating process can affect or be affected by other processes, including sharing data**
- **Reasons for cooperation:**
  - **Information sharing**
  - **Computation speedup**
  - **Modularity**
  - **Convenience**
- **Cooperating processes need *interprocess communication (IPC)***
  - **Message Passing**
  - **Shared memory**

ICS 212: Operating Systems

Lecture 7

Instructor:

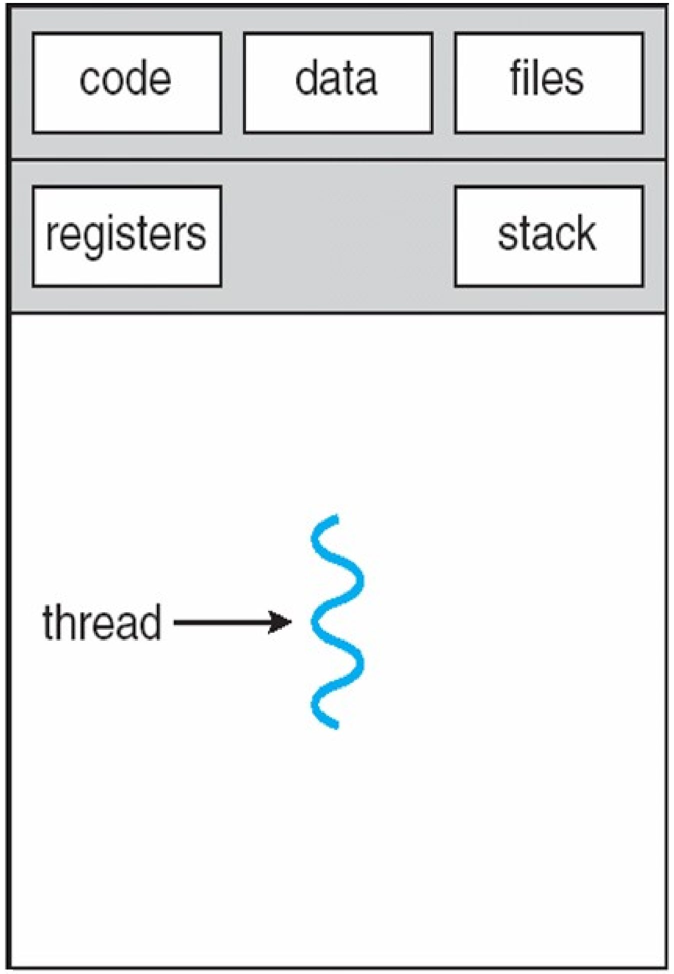
Dr. Nilotpal Chakraborty

Indian Institute of Information Technology Kottayam

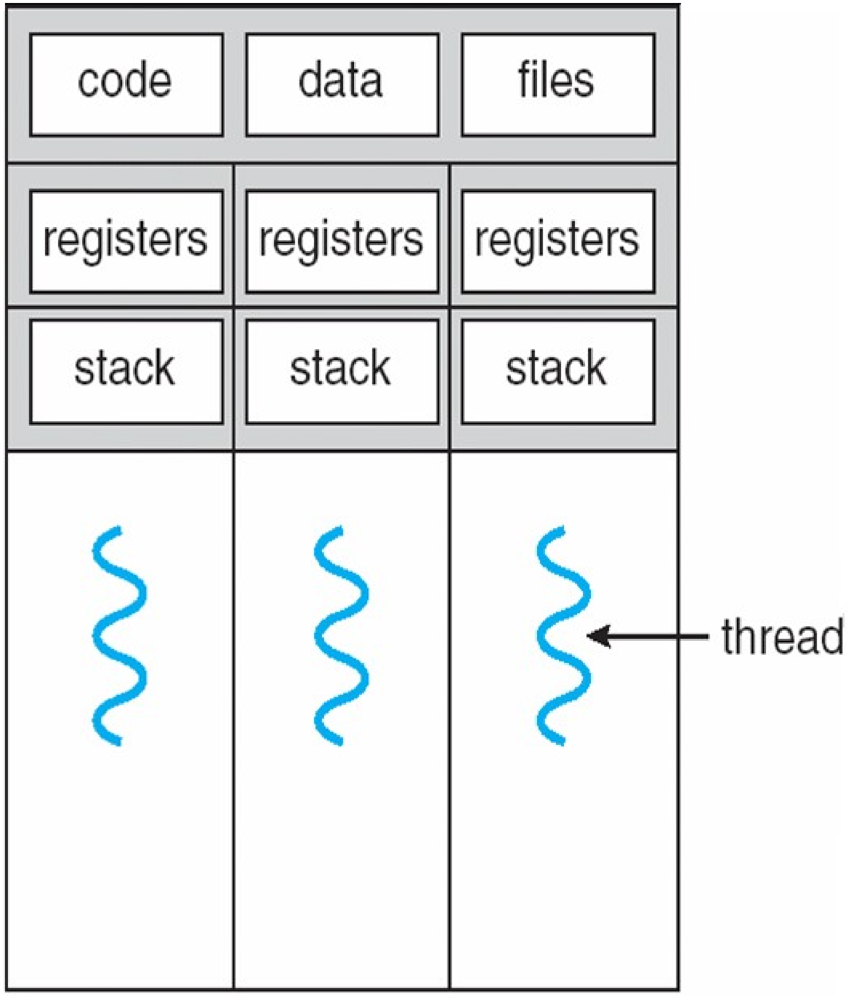
Threads

- A single sequential execution stream within a process.
- An ***independent unit of execution*** created ***within*** the context of a ***process***.
- ***Multithreading*** is a model of program execution that allows for *multiple threads to be created within a process*, executing independently but concurrently sharing process resources.

Single threading v Multi threading



single-threaded process



multithreaded process

Why threading?

- Responsiveness
- Resource Sharing
- Economy
- Scalability

ICS 212: Operating Systems

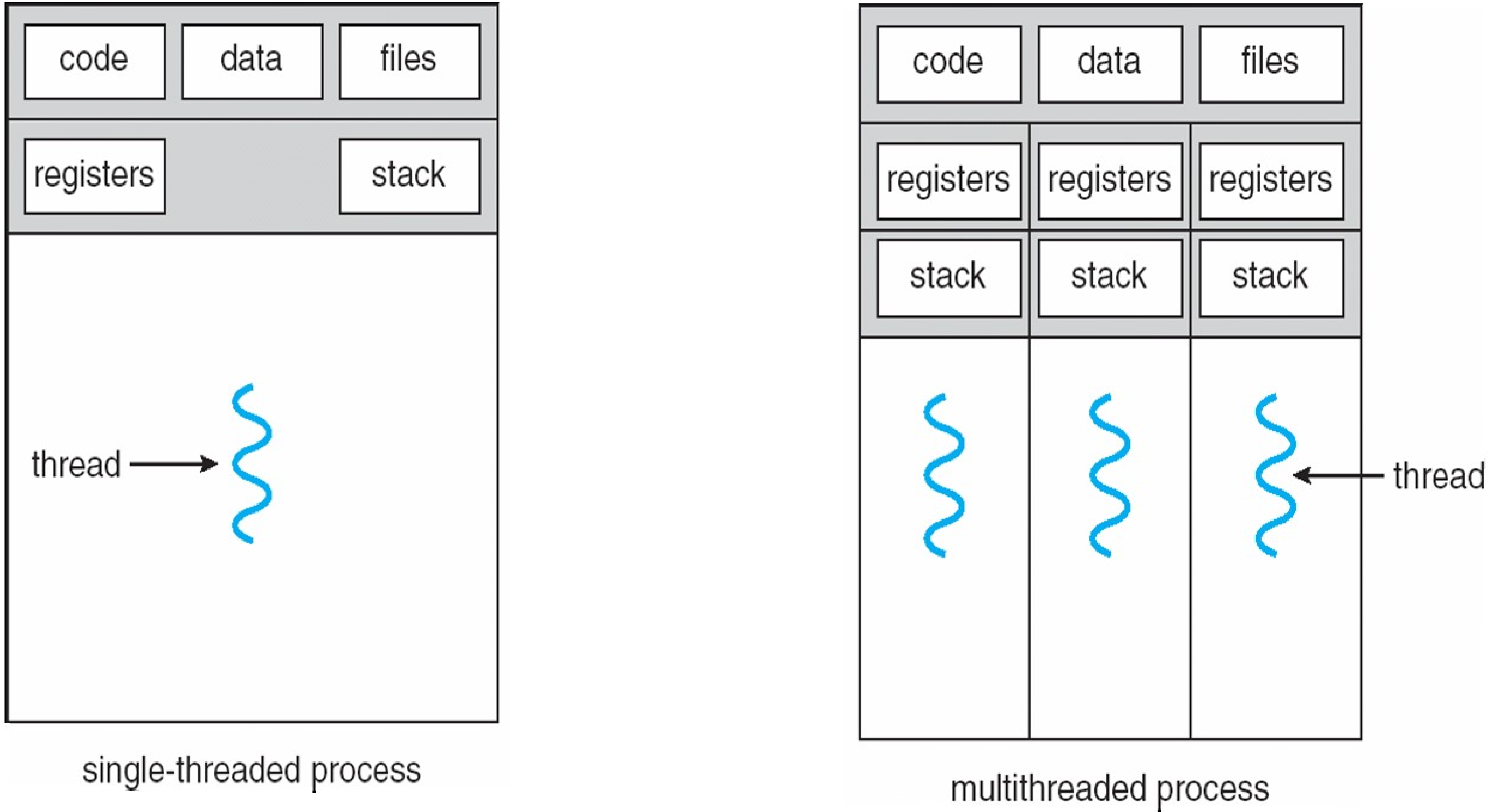
Lecture 8

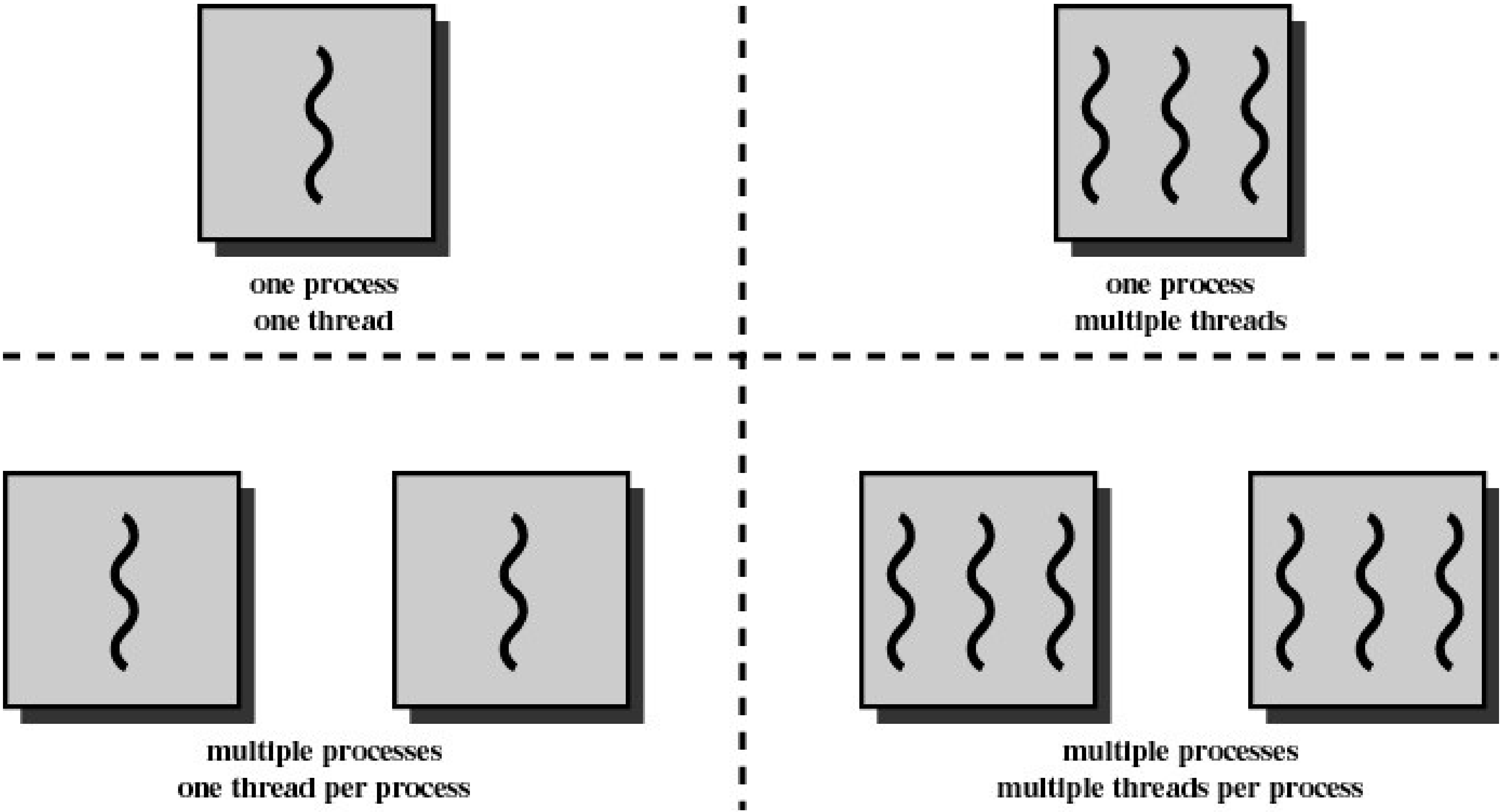
Instructor:

Dr. Nilotpal Chakraborty

Indian Institute of Information Technology Kottayam

Processes and Threads




one process
one thread
one process
multiple threads
multiple processes
one thread per process
multiple processes
multiple threads per process

Types of threads

- User-level threads
  - *All thread management is done by the application*
  - The kernel is not aware of the existence of threads
- Advantages of user-level threads −
  - User-level threads are easier and faster to create than kernel-level threads.
  - They are more easily managed.
  - User-level threads can be run on any operating system.
  - There are no kernel mode privileges required for thread switching in user-level threads.

Types of threads

- **Kernel-level threads**
  - ***Managed by the kernel***
  - **Scheduling is done on a thread basis**
- **Advantages of kernel-level threads:**
  - **Multiple threads of the same process can be scheduled on different processors in kernel-level threads.**
  - **The kernel routines can also be multithreaded.**
  - **If a kernel-level thread is blocked, another thread of the same process can be scheduled by the kernel.**

ICS 212: Operating Systems

Lecture 9

Instructor:

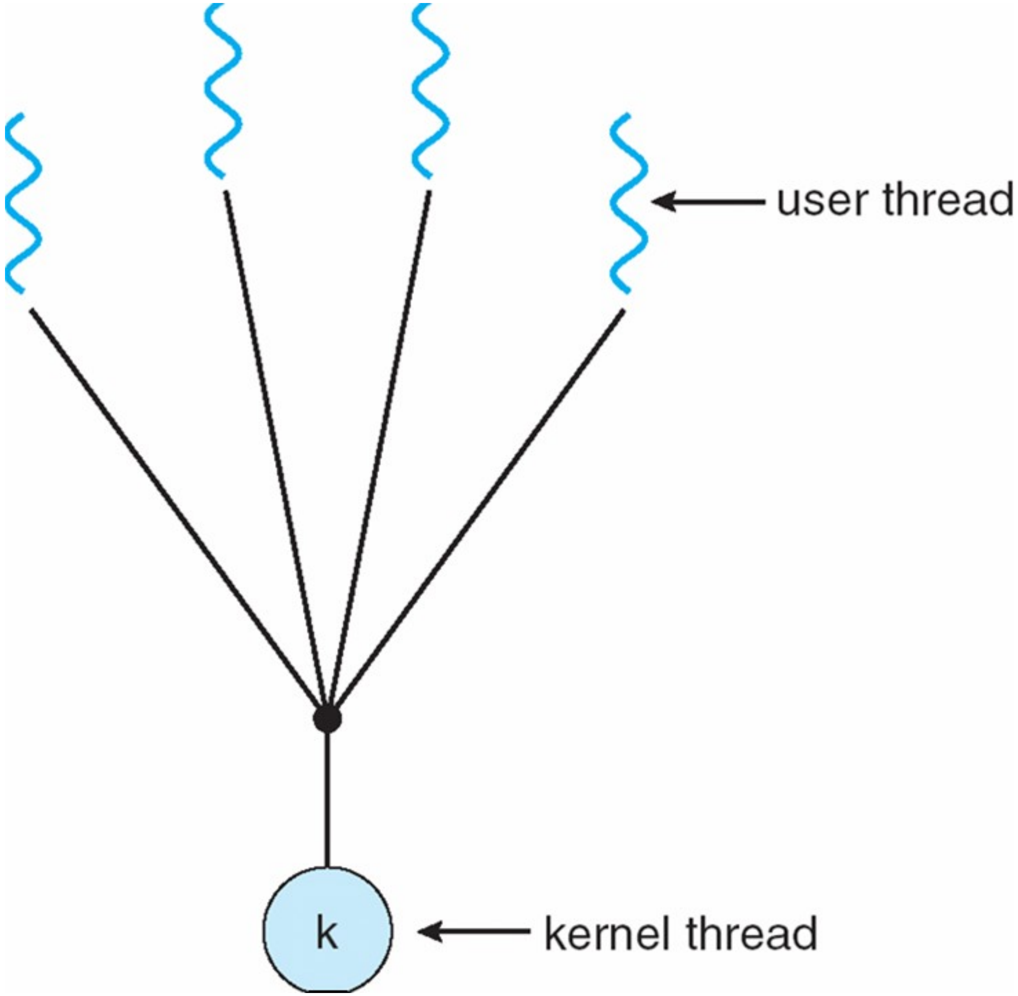
Dr. Nilotpal Chakraborty

Indian Institute of Information Technology Kottayam

Types of threads

- User-level threads
  - All thread management is done by the application
  - The kernel is not aware of the existence of threads
  - Implemented as a ***thread library*** which contains the code for thread creation, termination, scheduling and switching
- Kernel-level threads
  - Managed by the kernel
  - Scheduling is done on a thread basis

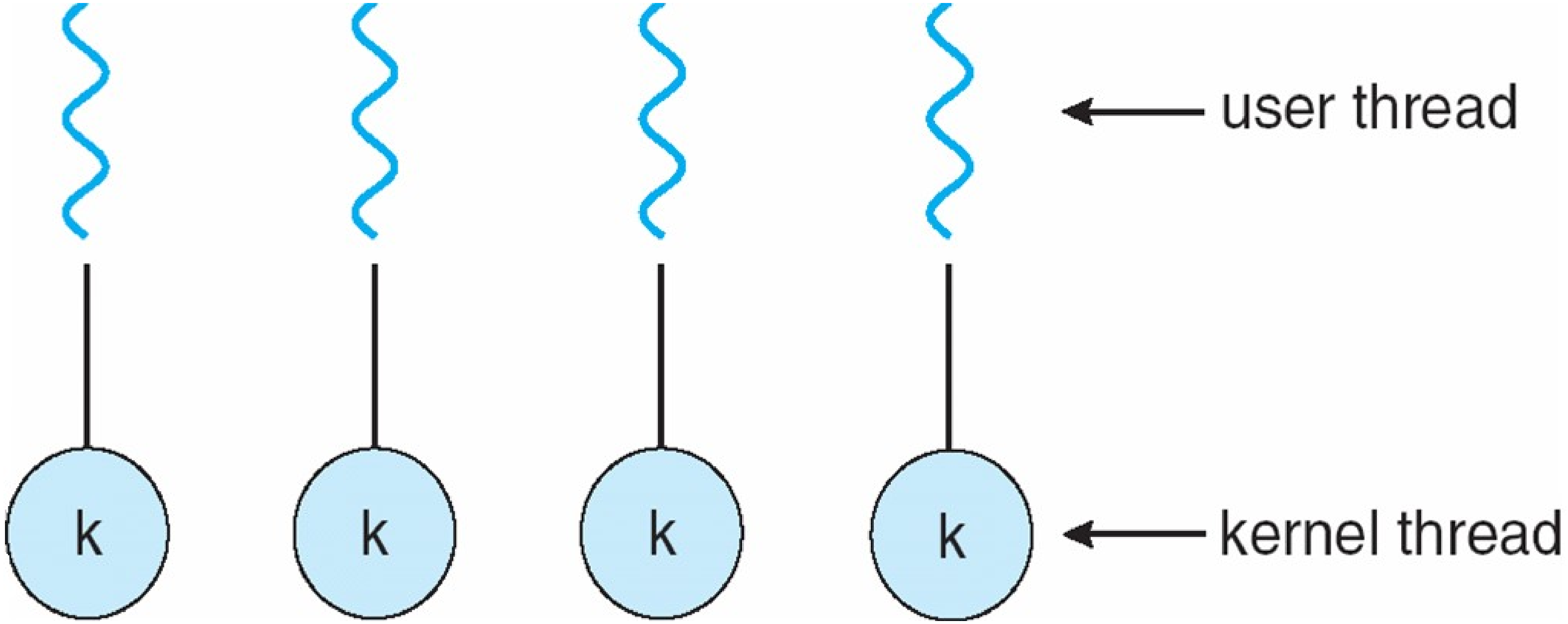
Many-to-one model



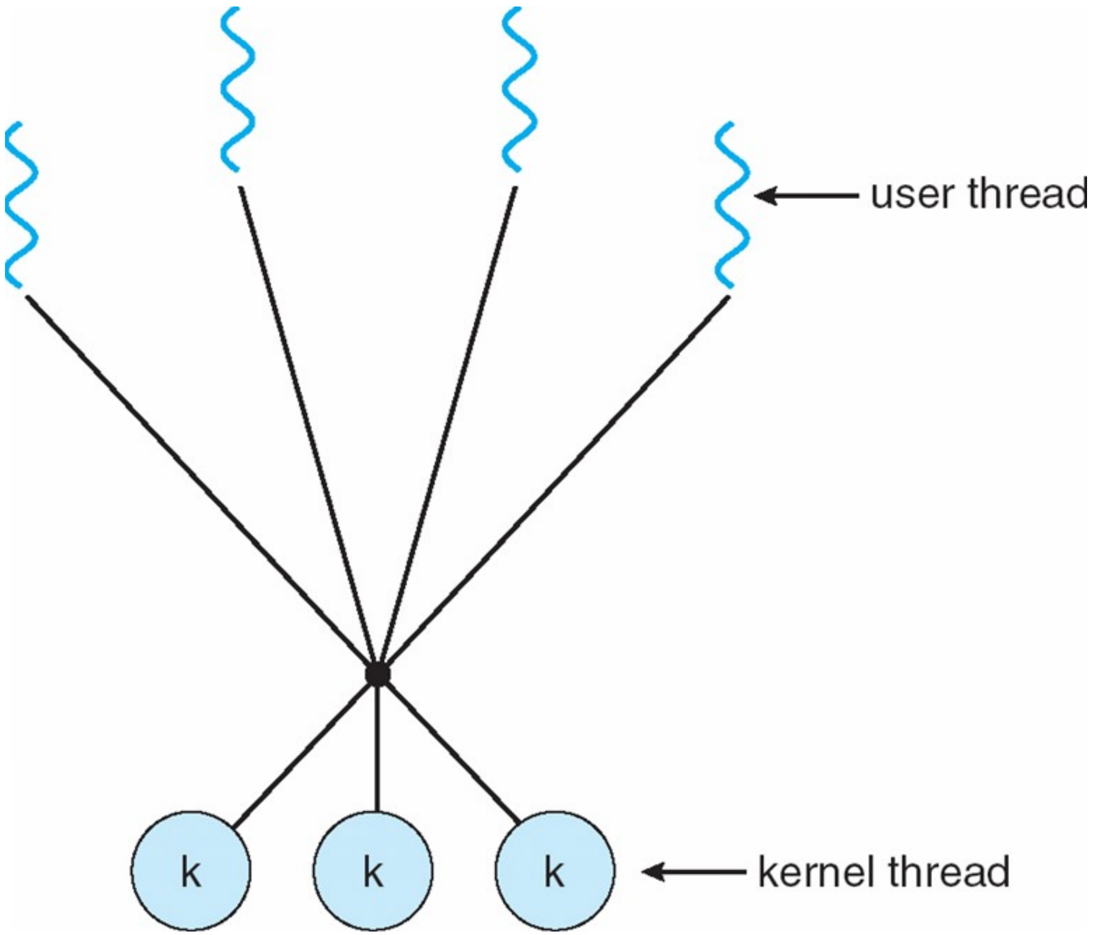
- Many user threads are mapped to one kernel thread
- The entire process will be ***blocked*** if a thread makes a blocking system call
- Multiple threads are unable to run in parallel

One-to-one model

- Each user thread is mapped to a kernel thread
- Multiple *threads can run in parallel* on multiprocessors
- Creating a user thread requires creating a corresponding kernel thread
- Implementational restriction

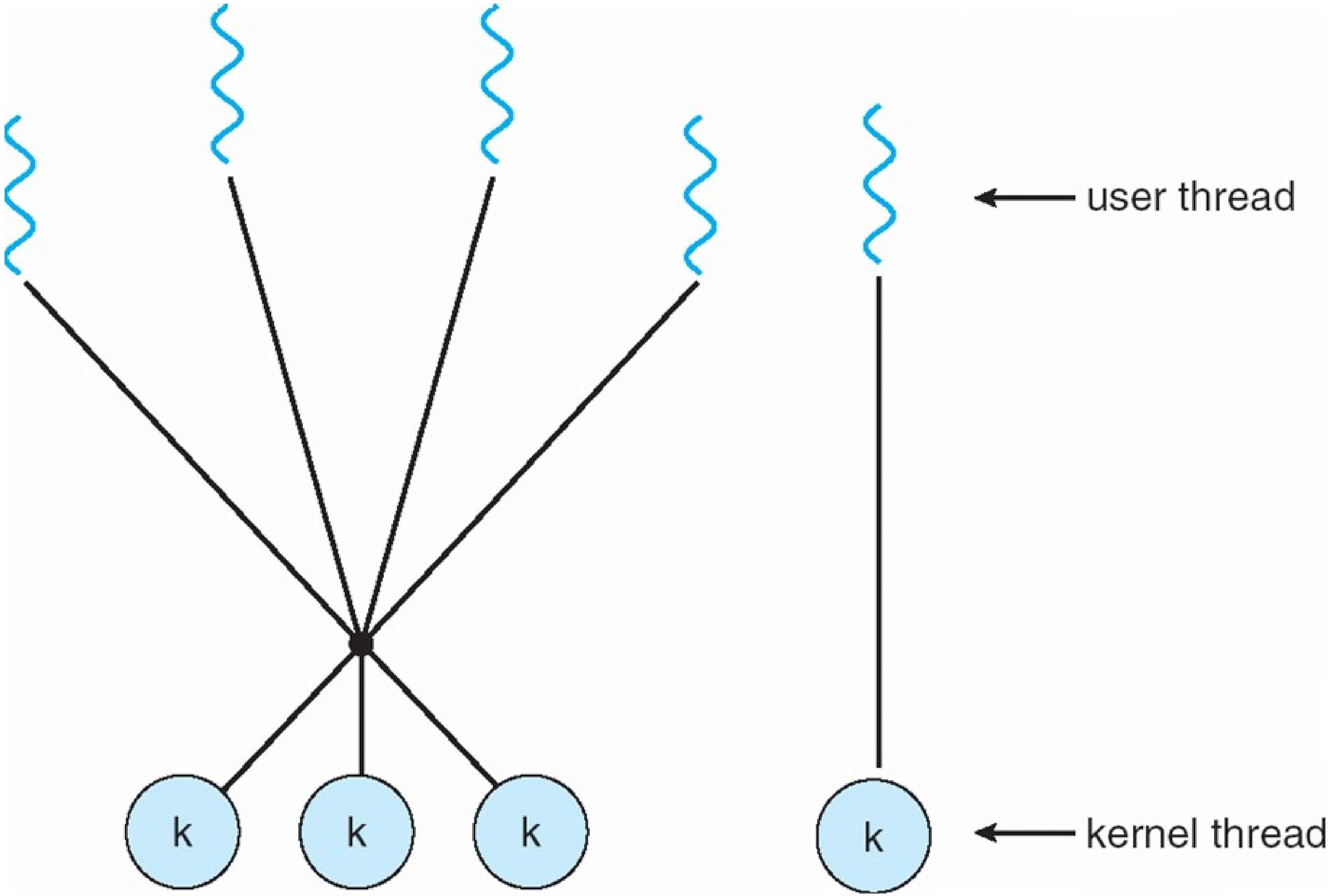


Many-to-many model



- Allows many user level threads to be mapped to some (equal or lesser) kernel threads
- Allows the *operating system to create a sufficient number of kernel threads*
- During a *blocking system call by one thread, kernel schedules another thread* for execution.

Two-level model



ICS 212: Operating Systems

Lecture 10

Instructor:

Dr. Nilotpal Chakraborty

Indian Institute of Information Technology Kottayam

Thread library

- Thread library provides programmers with APIs for creating and managing threads
- Two primary ways of implementing
  - Library entirely in user space
  - Kernel-level library supported by the OS

Common thread libraries

- POSIX threads:
  - A *POSIX* standard (IEEE 1003.1c) API
  - *Common in UNIX* based operating systems
  - Can be provided as *either a user level or a kernel level* library
- Win 32 threads
  - *For Windows* systems
  - *Only kernel* level library
- Java threads
  - Managed by the JVM
  - Typically implemented using the threads model provided by underlying OS as Java program

POSIX: Layers of Abstraction

- A ***Pthread*** is the POSIX specification for thread behaviour.
- A *kernel thread* is the entity actually created, scheduled and managed by the kernel as the context to execute a thread in a process
- *"The thread library may provide no more than a simple wrapper routine to a kernel system call to implement the POSIX specification; or it may provide full support for thread creation, scheduling and management, independently of the kernel."*

Pthread objects

Type	Description
pthread_t	Thread identifier
Pthread_attr_t	Thread attribute
Pthread_cond_t	Condition variable
Pthread_condattr_t	Condition variable attribute
Pthread_key_t	Access key for thread-specific data
Pthread_mutex_t	mutex
Pthread_mutexattr_t	Mutex attribute
Pthread_once_t	One time initialization
Pthread_rwlock_t	Read-write lock
Pthread_rwlockattr_t	Read-write lock attribute

Thread management functions in POSIX

Function	Description
Pthread_create	Create a thread
Pthread_cancel	Terminate another thread
Pthread_detach	Set thread to release resources
Pthread_equal	Test two thread IDs for equality
Pthread_exit	Exit a thread w/o exiting process
Pthread_kill	Send a signal to a thread
Pthread_join	Wait for a thread

ICS 212: Operating Systems

Lecture 11

Instructor:

Dr. Nilotpal Chakraborty

Indian Institute of Information Technology Kottayam