OPERATING SYSTEMS

Lecture # 2

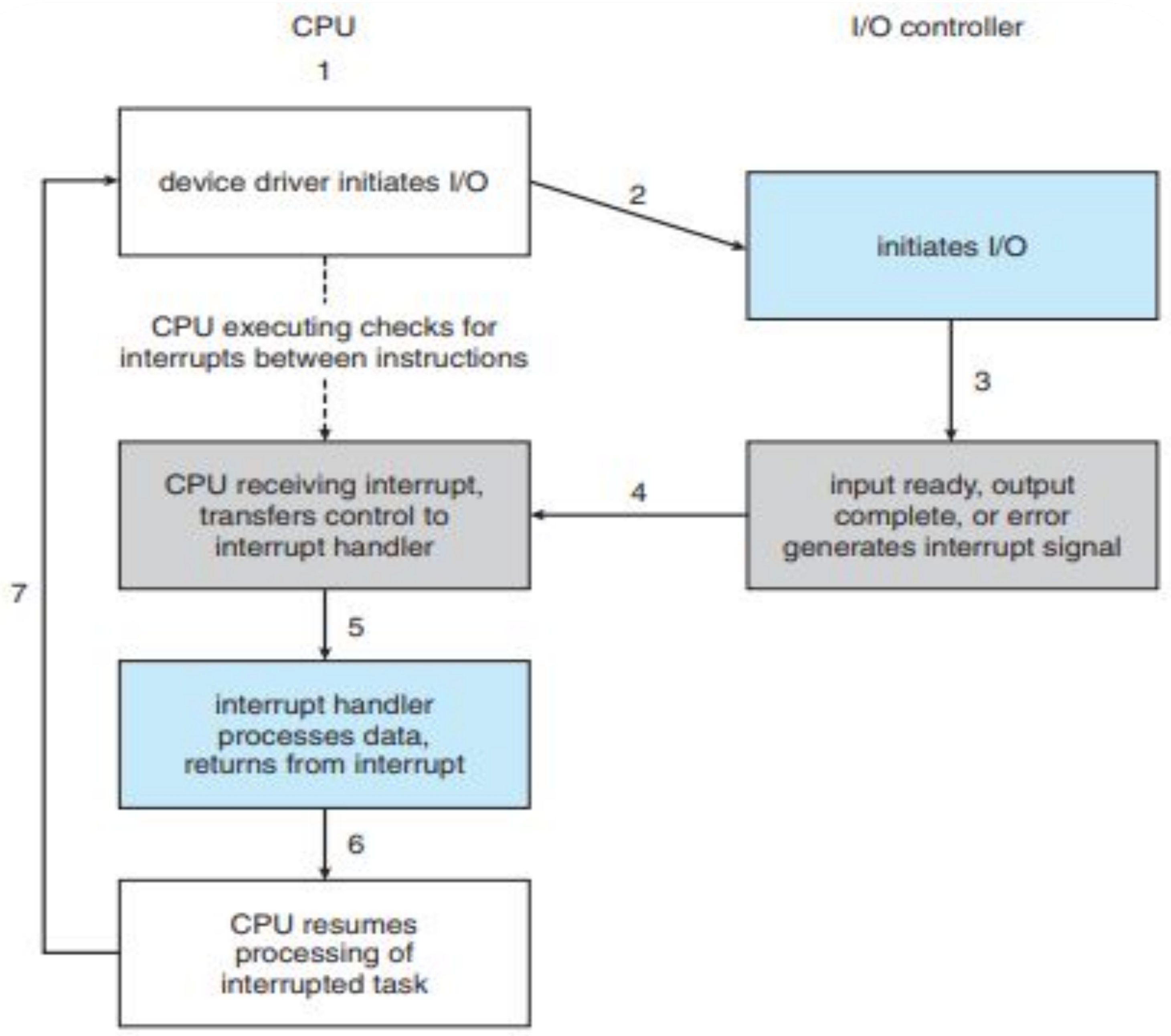
Razi Uddin

KERNEL

- The operating system is the one program running at all times on the computer—usually called the kernel.
- Along with the kernel, there are two other types of programs: system programs, which are associated with the operating system but are not necessarily part of the kernel,
- And application programs, which include all programs not associated with the operation of the system.
- The kernel is the core part of the operating system.
- The kernel has absolute control over all resources of the computer it is running on.
- The kernel is part of the OS that manages all resources.
- To ensure proper functioning of the system, the kernel is always memory resident.

INTERRUPTS

- An interrupt is a signal generated by a hardware device (usually an I/O device) to get CPU’s attention.
- Interrupt transfers control to the interrupt service routine (ISR), generally through the interrupt vector table, which contains the addresses of all the service routines.
- On completion the CPU resumes the interrupted computation.
- 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.
- An operating system is an interrupt-driven software.
- The operating system preserves the state of the CPU by storing registers and the program counter
- Determines which type of interrupt has occurred:
  - **polling**
  - **vectored** interrupt system


CPU
I/O controller
1
device driver initiates I/O
2
initiates I/O
CPU executing checks for interrupts between instructions
3
CPU receiving interrupt, transfers control to interrupt handler
4
input ready, output complete, or error generates interrupt signal
7
5
interrupt handler processes data, returns from interrupt
6
CPU resumes processing of interrupted task

TRAPS AND SIGNAL

- A trap (or an exception) is a software-generated interrupt caused either by an error (division by zero or invalid memory access) or by a user request for an operating system service.
- A signal is an event generated to get the attention of a process.
- An example of a signal is the event that is generated when you run a program and then press <Ctrl-C>. The signal generated in this case is called SIGINT (Interrupt signal).
- Three actions are possible on a signal:

1. Kernel-defined default action—which usually results in process termination and, in some cases, generation of a ‘core’ file that can be used by the programmer/user to know the state of the process at the time of its termination.
2. The process can intercept the signal and ignore it.
3. The process can intercept the signal and take a programmer-defined action.

HARDWARE PROTECTION

- Multi-programming put several programs in memory at the same time.
- While this increased system utilization it also increased problems.
- With sharing, many processes, could be adversely affected by a bug in one program.
- One erroneous program could also modify the program or data of another program or even the resident part of the operating system.
- A file may overwrite another file or folder on disk.
- A process may get the CPU and never relinquish it.
- So the issues of hardware protection are:

1. I/O protection,
2. memory protection,
3. and CPU protection

DUAL MODE OPERATION

- We must protect the operating system and all other programs and their data from any malfunctioning program.
- Protection is needed for any shared resources.
- Modern CPU has two kinds of instructions, privileged instructions and non-privileged instructions.
- Privileged instructions can be used to perform hardware operations that a normal user process should not be able to perform, such as communicating with I/O devices.
- If a user process tries to execute a privileged instruction, a trap should be generated.
- Operating system code should be allowed to execute privileged instructions.

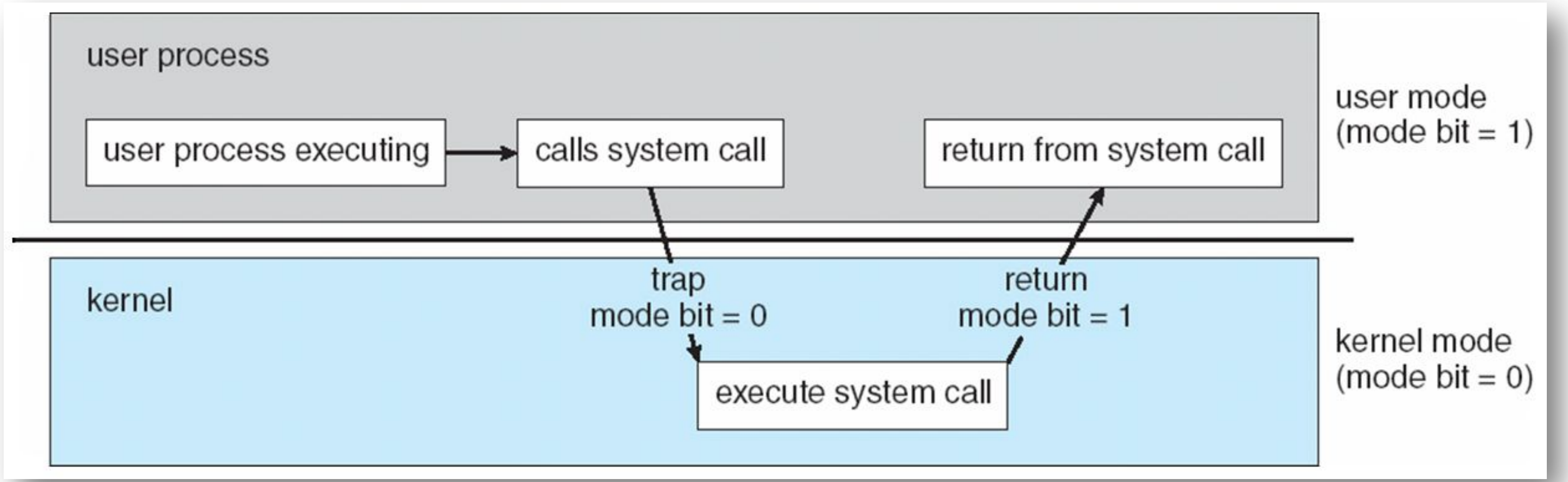
DUAL MODE OPERATION

- In order for the CPU to be able to differentiate between a user process and an operating system code, we need two separate modes of operation: user mode and monitor mode (also called supervisor mode, system mode, or privileged mode).
- **Mode bit** provided by hardware
- monitor mode (0) or user mode (1)
  - ✔ Provides the ability to distinguish when system is running user code or kernel code
  - ✔ Some instructions designated as **privileged**, only executable in kernel mode
  - ✔ System call changes mode to kernel, return from call resets it to user

DUAL MODE OPERATION

- The concept of privileged instructions also provides us with the means for the user to interact with the operating system by asking it to perform some designated tasks that only the operating system should do.
- A user process can request the operating system to perform such tasks for it by executing a system call.
- Whenever a system call is made or an interrupt, trap, or signal is generated.
- Mode is switched.

TRANSITION FROM USER TO KERNEL MODE



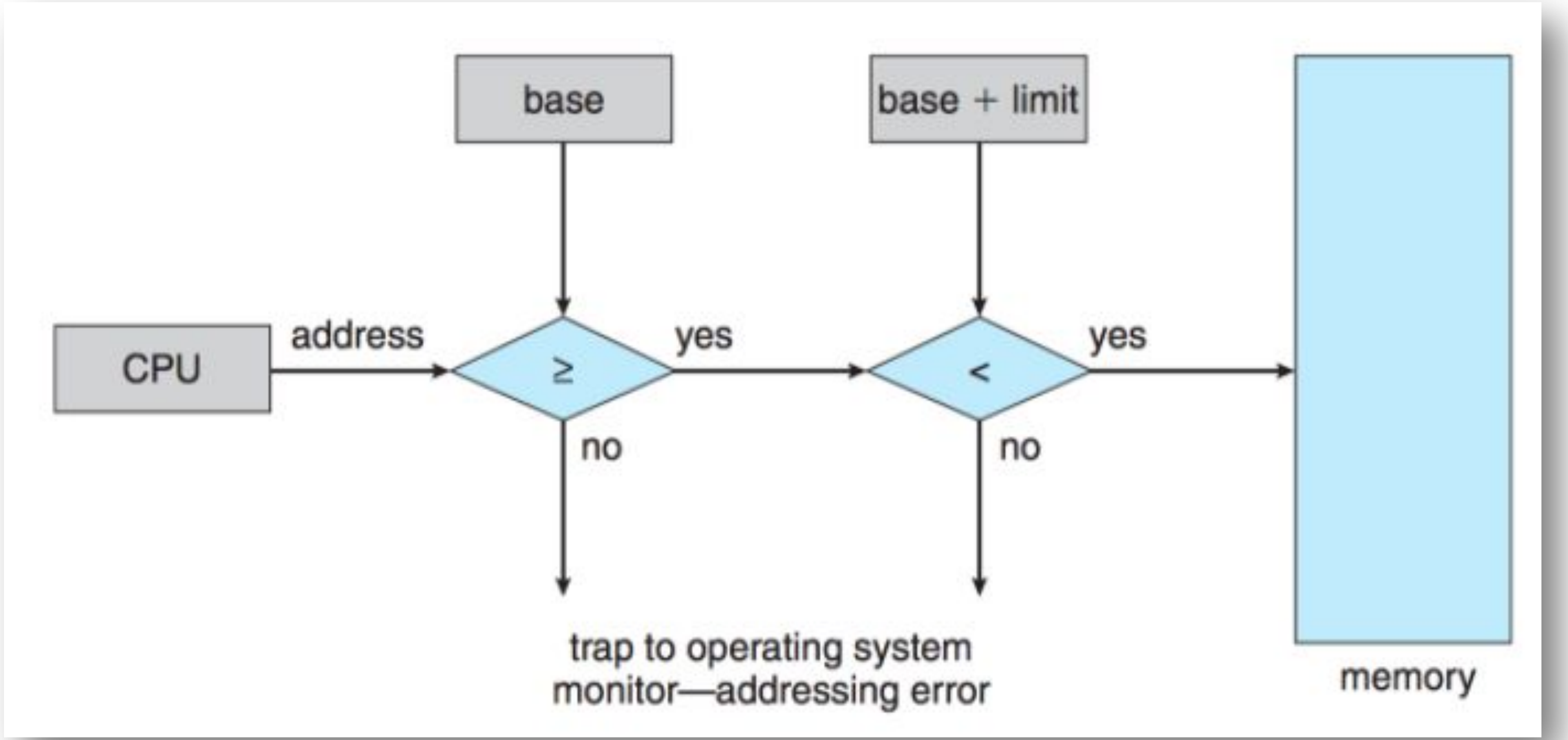
I/O PROTECTION

- A user process may disrupt the normal operation of the system by issuing illegal I/O instructions, by accessing memory locations within the operating system itself, or by refusing to relinquish the CPU.
- To prevent users from performing illegal I/O
- Define all I/O instructions to be privileged instructions.
- Thus users cannot issue I/O instructions directly; they must do it through the operating system.
- For I/O protection to be complete, we must be sure that a user program can never gain control of the computer in monitor mode.
- If it could, I/O protection could be compromised.

MEMORY PROTECTION

- The region in the memory that a process is allowed to access is known as process address space.
- To ensure correct operation of a computer system, we need to ensure that a process cannot access memory outside its address space.
- If we don’t do this then a process may, accidentally or deliberately, overwrite the address space of another process or memory space belonging to the operating system (e.g., for the interrupt vector table).
- Two registers are used for this purpose:
- **Base register—**(initialized with the starting address of the process)
- **Limit register—**(initialized with its size)
- Memory outside the defined range is protected because the CPU checks that every address generated by the process falls within the memory range defined by the values stored in the base and limit registers.

MEMORY PROTECTION



CPU PROTECTION

- We must prevent the user program from getting stuck in an infinite loop or not calling system services and never returning control to the CPU.
- To accomplish this we can use a timer, which interrupts the CPU after specified period to ensure that the operating system maintains control.
- Timer period may be variable or fixed.
- A fixed-rate clock and a counter are used to implement a variable timer.
- The OS initializes the counter with a positive value.
- The counter is decremented every clock tick by the clock interrupt service routine.

CPU PROTECTION

- When the counter reaches the value 0, a timer interrupt is generated that transfers control from the current process to the next scheduled process.
- Thus we can use the timer to prevent a program from running too long.
- In the most straightforward case, the timer could be set to interrupt every N milliseconds, where N is the time slice that each process is allowed to execute before the next process gets control of the CPU.
- The OS is invoked at the end of each time slice to perform various housekeeping tasks.