Operating Systems, Communications and Concurrency

Process Management Concepts

1. Programs, Processes, Processors

2. Representing Process Abstractions

3. Process Lifecycle

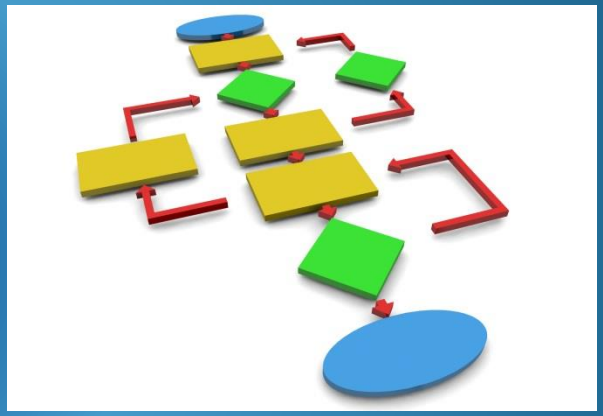
4. System Calls - Communicating with the Operating System

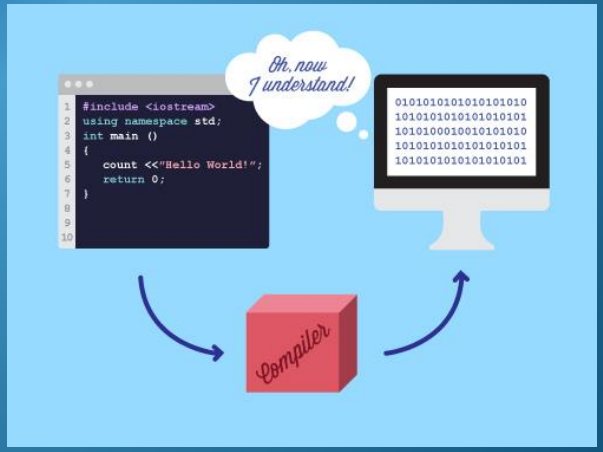
Operating Systems, Communications and Concurrency

1. Programs, Processes, Processors

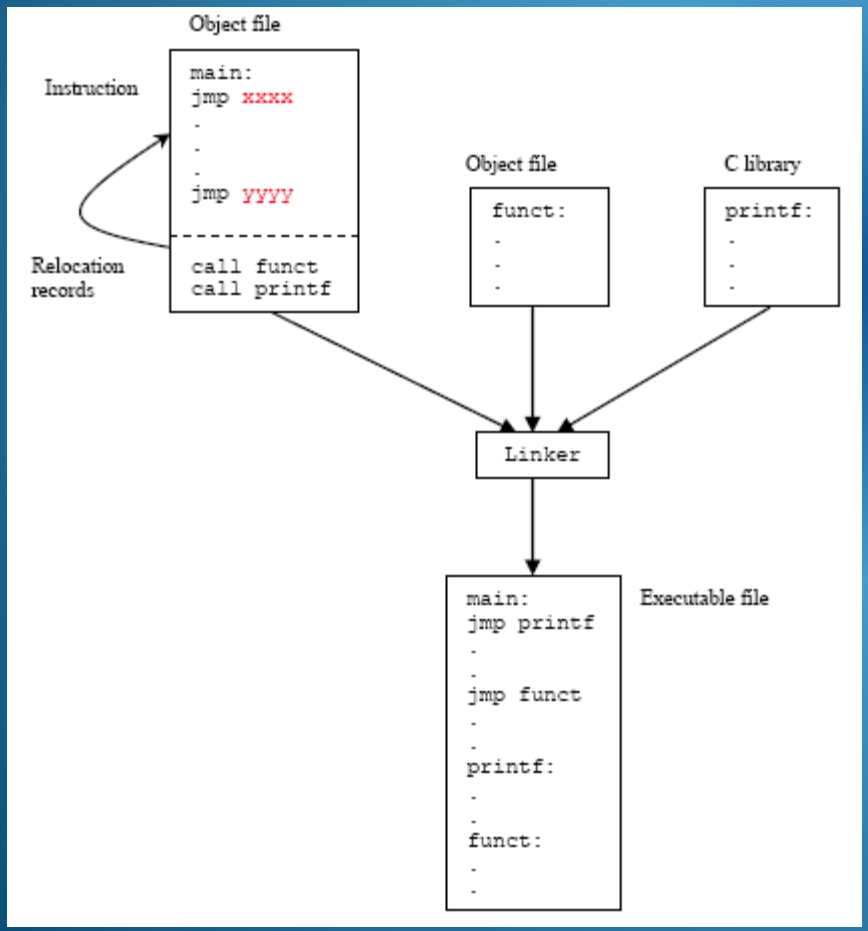
A **program** is a collection of instructions specifying a defined sequence of execution. It is the translation of an algorithm into a programming language.

A compiler maps the program code into a series of machine instructions for a particular processor and these are stored in an object file.





Operating Systems, Communications and Concurrency



Creating an executable file from a collection of compiled program objects.

Operating Systems, Communications and Concurrency

1. Programs, Processes, Processors

A **process** is an instance of a program in action. It is an operating system abstraction.

When the instructions are being carried out, a process exists.

Executable module of program is loaded as a program image in main memory and processor fetches instructions from it.

The program image has a format specific to a particular operating system and processor and includes any program parameters, stack space, data space and program code.

Operating Systems, Communications and Concurrency

1. Programs, Processes, Processors

A process is an execution context, a collection of kernel managed information needed while it is running.

Processes are dynamic entities whose lifetimes range from a few milliseconds to maybe months.

Processes may be persistent, implementing system services.

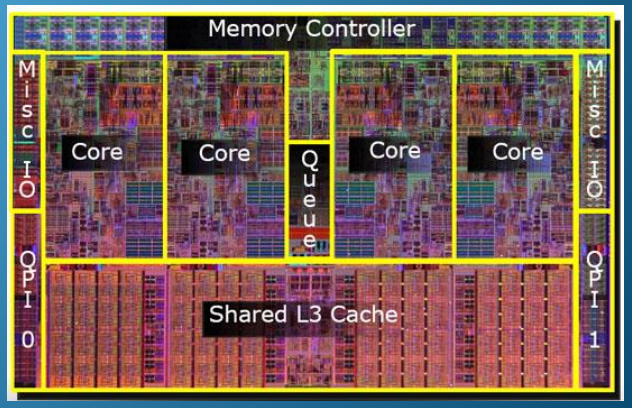
The operating system itself may be made up of several processes such as the process manager, the memory manager, the file manager and so on.

Operating Systems, Communications and Concurrency

1. Programs, Processes, Processors

A **processor** is the agent which runs a process by executing the instructions stored in the memory image.

In most desktop machines, there is only one processor (sometimes called CPU – central processing unit).



Modern processors have multiple execution "cores" and are capable of executing the instructions of more than one process simultaneously.

Operating Systems, Communications and Concurrency

1. Programs, Processes, Processors

Processor Time Sharing of Ready Processes

As there are far more processes than processors in a system, processes share the processor's resources by having the CPU's time divided amongst the ready processes when they have instructions to be executed.

Operating Systems, Communications and Concurrency

1. Programs, Processes, Processors

Blocked Processes

Processes are not always ready to use the processor as sometimes they must wait for other devices that are perhaps supplying data to them or they must wait for resources that need to be assigned, or are already in use, or perhaps they must wait for user interaction or time periods that must occur before they can continue.

Operating Systems, Communications and Concurrency

2. Representing Process Abstractions

A fundamental task of an operating system is process management – Creating, Controlling, Terminating – Managing the Execution Environment

The operating system must allocate resources such as memory space and cpu time to processes, it must protect resources from activities of other processes, and provide synchronisation mechanisms among processes that share the same resources.

Operating Systems, Communications and Concurrency

2. Representing Process Abstractions

To achieve this fundamental task, an operating system uses data structures to represent the state of process objects it is dealing with.

For each process, the operating system maintains a process control block (PCB) or process descriptor to keep a clear picture of what each process is doing, what point it has reached in its execution and what resources have been assigned to it.

Operating Systems, Communications and Concurrency

2. Representing Process Abstractions

A Process Control Block is used to keep track of the execution context (all resource information about the process and its activity) that can be used for independent scheduling of that process onto any available processor.

Process Identification Data

Processor State

Process Control Data

Operating Systems, Communications and Concurrency

2. Representing Process Abstractions

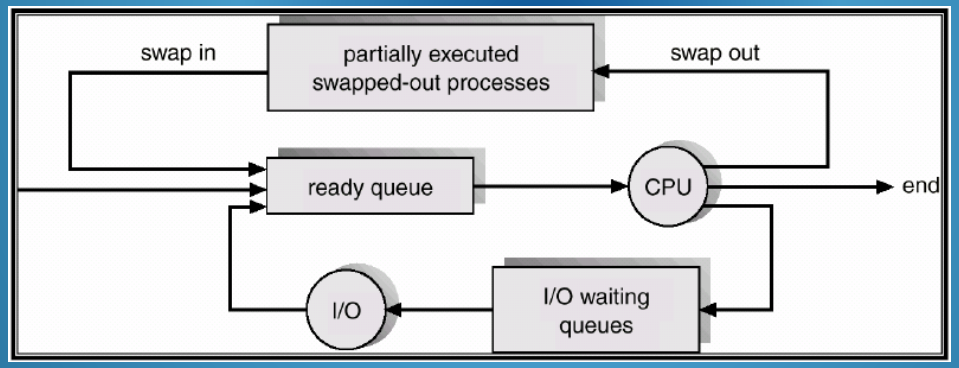
Process Identification Data	Processor State Data	Process Control Data
Unique process identifier	CPU Register State,	Flags, signals and messages.
Owner's user identifier	Pointers to stack and memory space	Pointers to other processes in the same queue
Group identifier	Priority	Parent and Child linkage pointers
	Scheduling Parameters	Access permissions to I/O Objects
	Events awaited	Accounting Information

Simplified Process Control Block (PCB)

Operating Systems, Communications and Concurrency

3. Process Lifecycle

The PCB may be moved between different queues over the process lifetime depending on the priority or state of its execution.

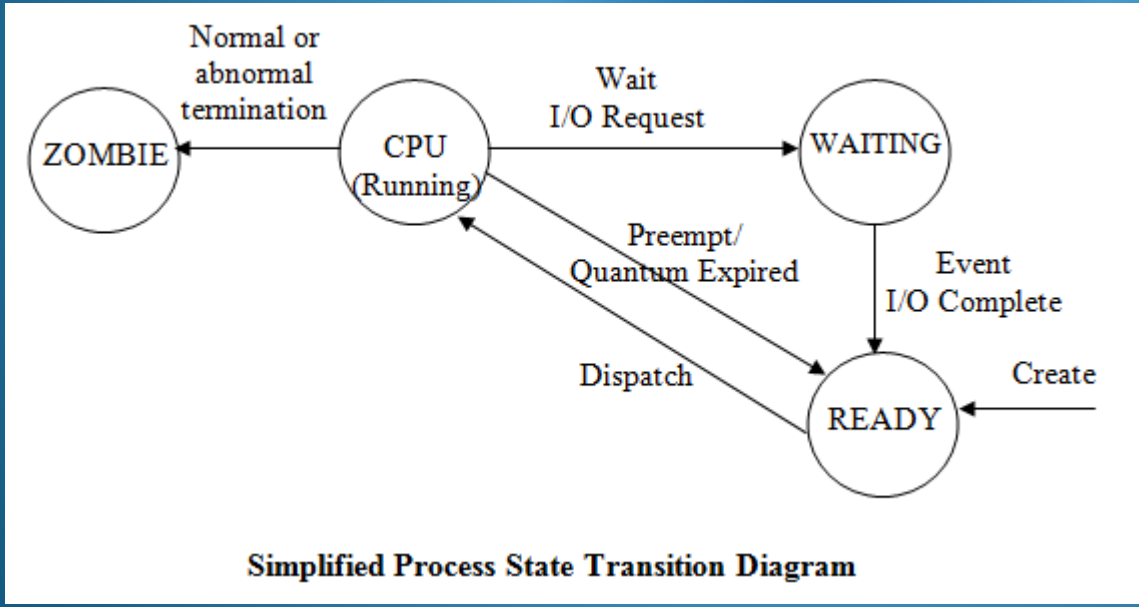


Queues hold processes waiting for different resources which are serviced using scheduling algorithms.

The Process Control Block of a process represents the process on one of the OS queues.

Operating Systems, Communications and Concurrency

3. Process Life Cycle



Simplified Process State Transition Diagram

There will be periods where it needs the CPU to execute its instructions and there will be other periods where it is waiting on various other system resources, such as input/output devices, to provide data so that it can continue its execution.

Operating Systems, Communications and Concurrency

4. Communicating with the OS

For security and reliability reasons, direct communication or interference between unrelated processes or with the hardware and operating system itself can only be achieved by using a specific mechanism.

Communication with other processes in the system is regulated by using interprocess communication functions provided by the OS. These functions map the message data from one process to the other.

Communication with the operating system is done via a special system call mechanism which automatically switches the mode of execution of the processor.

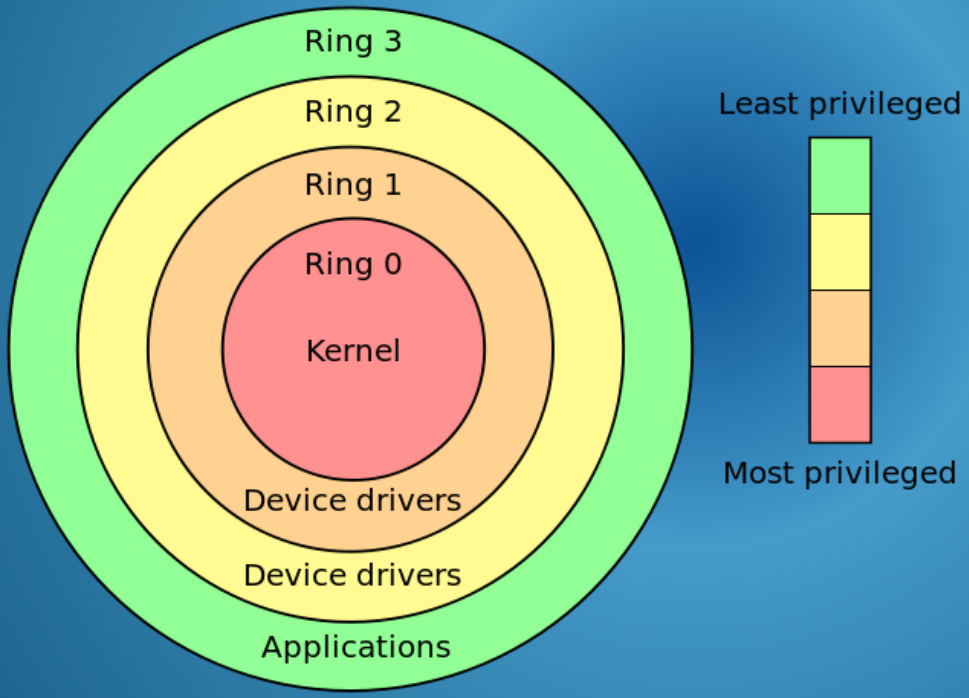
Operating Systems, Communications and Concurrency

4. Communicating with the OS

In order to enforce hardware protection, certain processor instructions are restricted and cannot be executed by ordinary user processes.

The processor executes in one of two modes, User Mode or Supervisor Mode. When executing a user process the processer is in User Mode, and can only execute a subset of its instruction set. To execute operating system code, the processor must be switched to Supervisor Mode in a controlled manner and can then execute its full instruction set.

Operating Systems, Communications and Concurrency



Processor Modes of Execution

Processors may implement several modes of execution but require at least two modes to protect the system from user process activities.

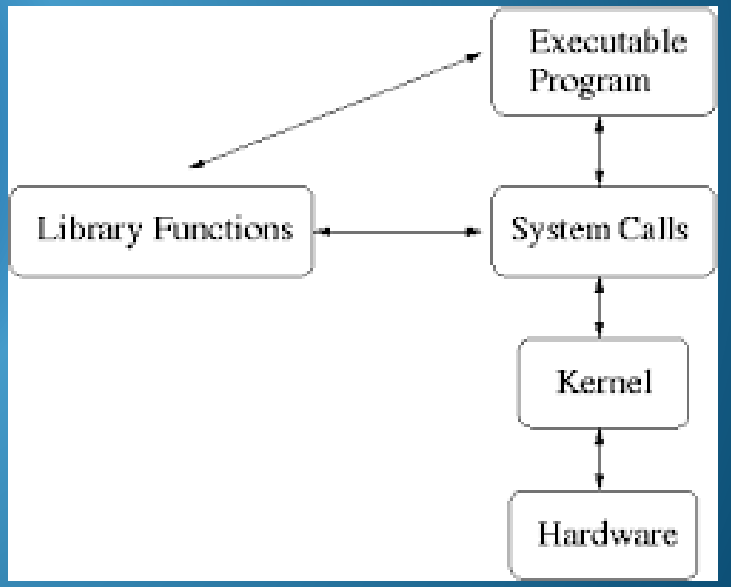
OS code executes in a privileged mode where a wider instruction set is available in the processor, instructions that are not available in user mode.

Operating Systems, Communications and Concurrency

4. Communicating with the OS

Switching to Supervisor Mode

A special processor instruction known as a **software interrupt** is the mechanism for doing this.



Processes need to communicate with the operating system in order to obtain protected system services like accessing the hard disk or other hardware, creating new processes, doing interprocess communication or configuring kernel services.

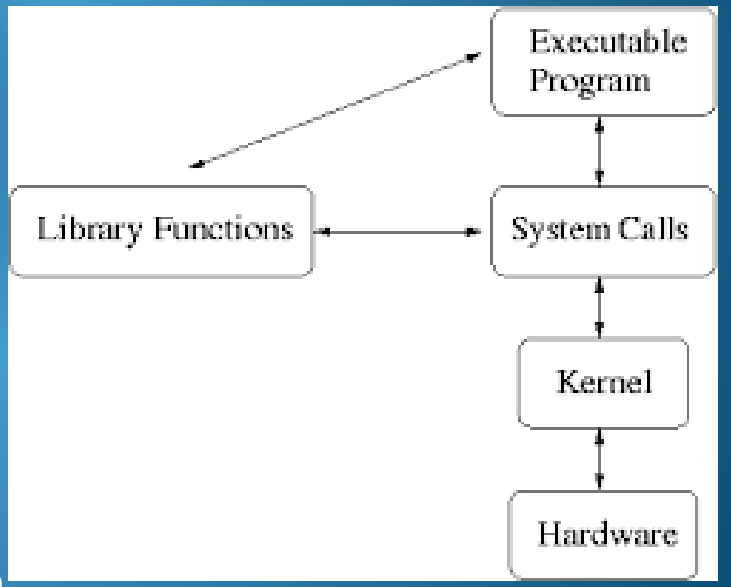
Operating Systems, Communications and Concurrency

4. Communicating with the OS

System Calls are often accessed through wrapper libraries linked with the user space process.

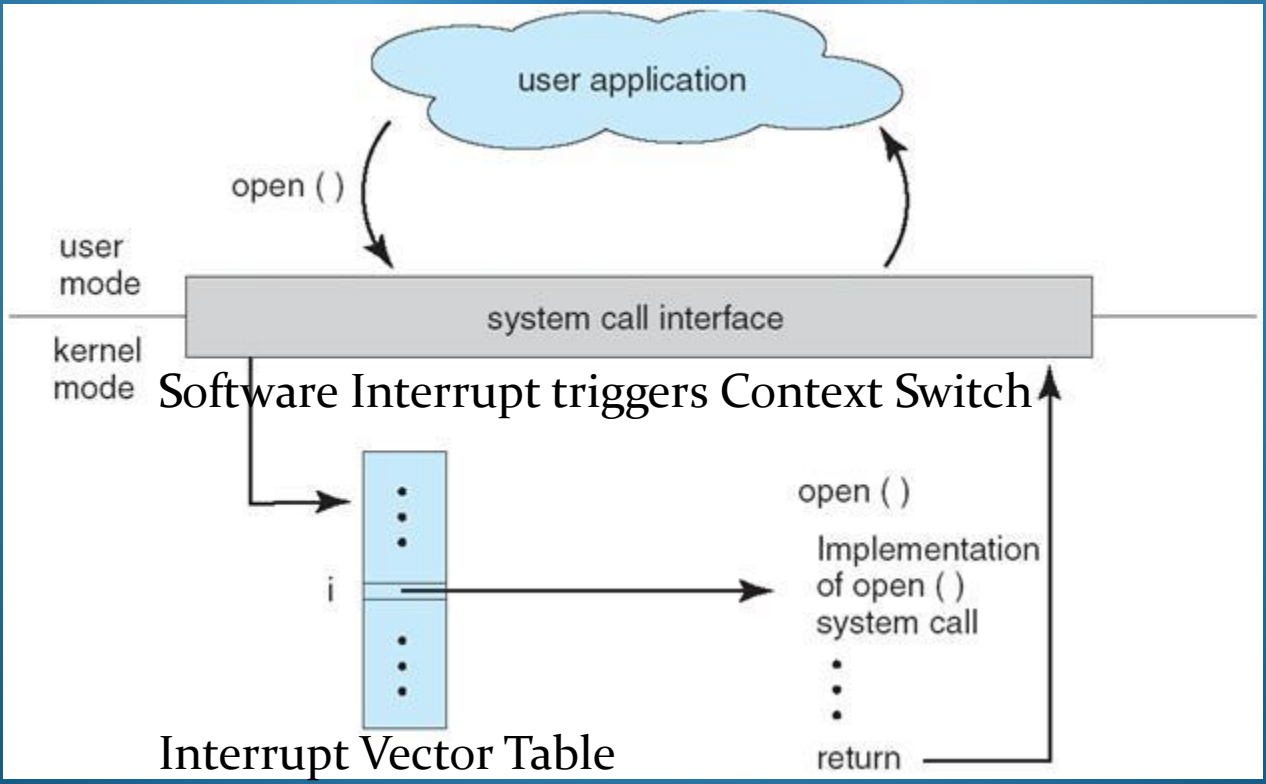
The wrapper function checks the correct parameters are used and will invoke the software interrupt mechanism.

A wrapper API for system calls serves to help application code portability across different kernel and language implementations.



Operating Systems, Communications and Concurrency

4. Communicating with the OS



Operating Systems, Communications and Concurrency

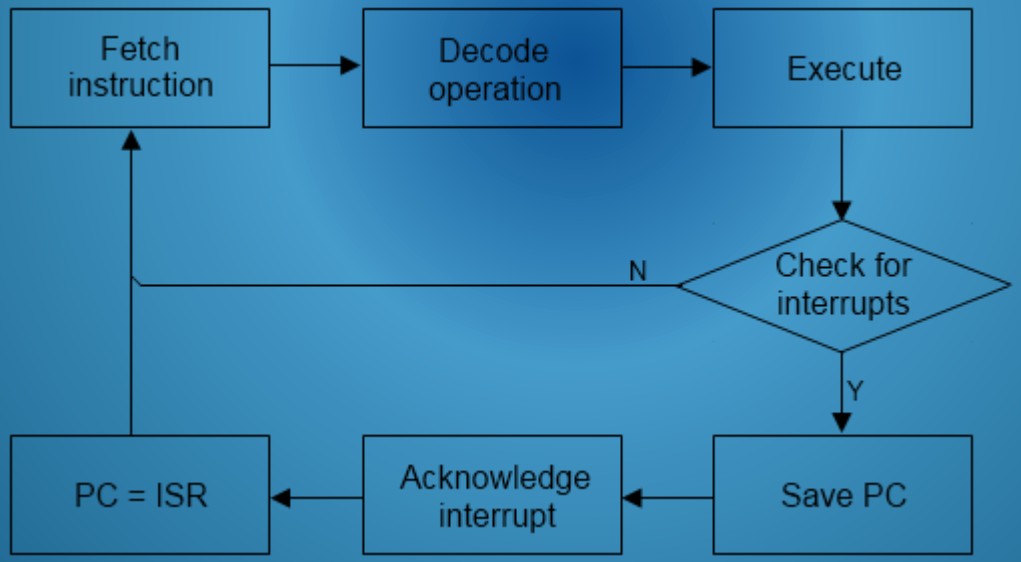
Hardware Communicating with the OS

This is achieved using **Hardware Interrupt** mechanisms.

Electrical signals are sent from hardware devices to indicate that they need attention or have completed a task.

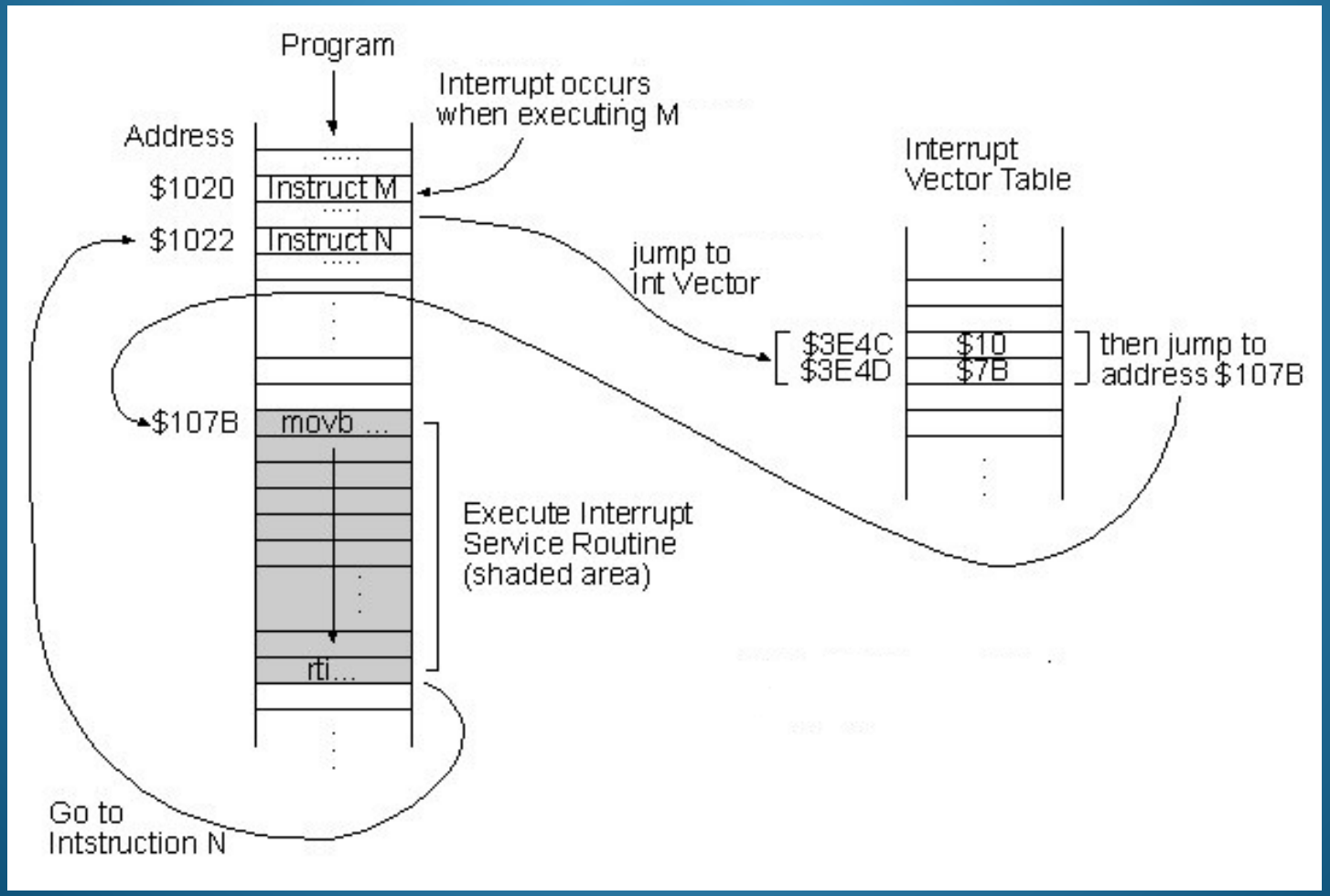
Operating Systems, Communications and Concurrency

4. Communicating with the OS



Interrupts Detected during Instruction Execution Cycle

Operating Systems, Communications and Concurrency



Operating Systems, Communications and Concurrency

4. Communicating with the OS

The Hardware Interrupt mechanism is needed to enable the system to effectively **manage a large number of hardware devices efficiently**. Polling the status of devices is not efficient.

Usually a support chip is used to manage, field and identify interrupts and send them to the processor.

Operating Systems, Communications and Concurrency

4. Communicating with the OS

A Hardware Interrupt mechanism is also needed to implement a **multitasking environment**.

When a task is to be scheduled to use the processor for a set amount of time, a clock timer is initialised. When the time expires an interrupt signal invokes the operating system scheduler to select another process. This scheme prevents one process from hogging the CPU indefinitely.

Operating Systems, Communications and Concurrency

Summary

Programs, Processes, Processors

Process Control Blocks

Process Life Cycle

System Calls and Hardware Interrupts