System Programming

Lecture 1
System Programming: Introduction, Operating System Examples

Prepared by
Hazim Noman Abed
Hazim_numan@yahoo.com

System Programming: Introduction

- What is System?

  System is the collection of various components

  Ex:- College is a system

- What is Programming?

Art of designing and implementing the programs.

- What is Software ?

Software is collection of many programs

System Programming: Introduction

- Two types of software
  - System software

    These programs assist general use application programs

    Ex:- Operation System , Assembler etc.

  - Application software

    These are the software developed for the specific goal

- System Program:-

  "These are programs which are required for the effective execution of general user programs on computer system."

- System Programming:-

  " It is an art of designing and implementing system programs."

System Programming: Introduction

Components of System Programming

- Interpreter
- Assembler
- Compiler
- Macros and Microprocessors
- Formal systems
- Debugger
- Linkers
- Operating system

What is an Operating System?

- An OS is a program which acts as an interface between computer system users and the computer hardware.
- It provides a user-friendly environment in which a user may easily develop and execute programs.
- An operating system is a powerful, and usually large, program that controls and manages the hardware and other software on a computer.
- An operating system (OS) is software that manages computer hardware and software resources and provides common services for computer programs.
- The operating system is an essential component of the system software in a computer system. Application software usually require an operating system to function.

Basic Functions of the Operating System

Device configuration

- Controls peripheral devices connected to the computer

File management

- Transfers files between main memory and secondary storage, manages file folders, allocates the secondary storage space, and provides file protection and recovery

Memory management

- Allocates the use of random access memory (RAM) to requesting processes

Interface platform

- Allows the computer to run other applications

Other Functions of the Operating System

- Best use of the computer resources
- Provide a background for user's programs to execute
- Display and deal with errors when it happens
- Control the selection and operation of the peripherals
- Act as a communication link between users
- System protection

Operating System Examples (DOS)

- In the 1980s or early 1990s, the operating system that shipped with most PCs was a version of the *Disk Operating System (DOS)* created by Microsoft: *MS-DOS*.

- MS-DOS is a disk operating system for IBM PC–compatible computers.

- In its day, it was easily the most popular operating system in the world.

Operating System Examples (DOS)

- As with any other operating system, its function is to oversee the operation of the system by providing support for executing programs, controlling I/O devices, handling errors, and providing the user interface.
- MS-DOS is a single-user, single-task operating system. These qualities make it one of the easiest disk operating systems to understand.
- The main portions of MS-DOS are the *IO.SYS*, *MSDOS.SYS*, and *COMMAND.COM* files.
- IO.SYS and MSDOS.SYS are special, hidden system files

Operating System Examples (DOS)

- The IO.SYS file moves the system's basic I/O functions into memory and then implements the MS-DOS default control programs, referred to as device drivers, for various hardware components. **These include the following**:
  - The boot disk drive
  - The console display and keyboard
  - The system's time-of-day clock
  - The parallel and serial communications port

Operating System Examples (DOS)

- The COMMAND.COM command interpreter accepts commands issued through the keyboard, or other input device, and carries them out according to the commands definition

- When DOS runs an application, COMMAND.COM finds the program, loads it into memory, and then gives it control of the system. When the program is shut down, it passes control back to the command interpreter.

Operating System Examples(Microsoft Windows)

- Microsoft is an American multinational corporation headquartered in Redmond, Washington, that develops, manufactures, licenses, supports and sells computer software, consumer electronics and personal computers and services.

- The **history of Microsoft** began on April 4, 1975.

- when it was founded by Bill Gates and Paul Allen in Albuquerque.

- Its current best-selling products are the Microsoft Windows operating system and the Microsoft Office suite of productivity software.

Operating System Examples(Microsoft Windows)

- In 1980, Microsoft formed a partnership with IBM that allowed them to attach Microsoft's Operating system with IBM computers, paying Microsoft a royalty for every sale.
- In 1985, IBM requested that Microsoft write a new operating system for their computers called OS/2.
- When Microsoft launched several versions of Microsoft Windows in the 1990s, they had captured over 90% market share of the world's personal computers.
- The company has now become largely successful.

Operating System Examples(Microsoft Windows)

- Microsoft Windows is a series of graphical interface operating systems developed, marketed and sold by Microsoft.

- Microsoft introduced an operating environment named *Window* on November 20, 1985 as an add-on to MS-DOS in response to the growing interest in graphical user interfaces (GUIs).

- Microsoft Windows came to dominate the world's personal computer market with over 90% market share, overtaking Mac OS, which had been introduced in 1984.

- Latest Operating System Are – Window 8.

Operating System Examples(Macintosh)

- **Development:**
- 1979: The Macintosh project was started with Jef Raskin , who envisioned an easy-to-use, low-cost computer for the average consumer.

- September 1979: Raskin began looking for an engineer who could put together a prototype.

- Bill Atkinson, a member of the Apple Lisa Team, introduced him to Burrell Smith, a service technician who had been hired earlier that year.

Operating System Examples(Macintosh)

- **History**

- Lisa OS :
  - The original Macintosh system software was partially based on it, previously released by Apple for the Lisa computer in 1983.

- January 24, 1984
  - Apple Computer Inc. (now Apple Inc .) introduced the Macintosh personal computer with the Macintosh 128K model, which came bundled with what was later renamed the Mac OS Operating System .

- 1997 :
  - Apple marketed its operating system software as "Mac OS", beginning in 1997.

Operating System Examples(Macintosh)

- **Release**

- System :
  - is easily distinguished between other operating system from the same period because it does not use a command line interface.
  - it was one of the first operating systems to use an entirely graphical user interface .

Operating System Examples(UNIX)

- UNIX is a computer operating system, a control program that works with users to
  - run programs,
  - manage resources, and
  - communicate with other computer systems.

- Several people can use a UNIX computer at the same time; hence UNIX is called a multiuser system. Any of these users can also run multiple programs at the same time; hence UNIX is called multitasking.

- The philosophy behind the design of Unix was to provide simple, yet powerful utilities that could be pieced together in a flexible manner to perform a wide variety of tasks.

Operating Systems Examples(UNIX)

Main Features of UNIX:

- Multi-user: more than one user can use the machine at a time , supported via terminals (serial or network connection) .
- Multi-tasking: more than one program can be run at a time
- Hierarchical directory structure, to support the organisation and maintenance of files
- Portability: only the kernel (<10%) written in assembler. This meant the operating system could be easily converted to run on different hardware
- Tools for program development, a wide range of support tools (debuggers, compilers)

Operating Systems Examples (LINUX)

- Linux is one of popular version of UNIX operating System. It is open source as its source code is freely available. It is free to use. Linux was designed considering UNIX compatibility. It's functionality list is quite similar to that of UNIX.
- Components of Linux System

Basic Features

Following are some of the important features of Linux Operating System:

- Portable - Portability means softwares can works on different types of hardwares in sameway. Linux kernel and application programs supports their installation on any kind of hardware platform.

Operating Systems Examples (LINUX)

- **Open Source** - Linux source code is freely available and it is community based development project. Multiple teams works in collaboration to enhance the capability of Linux operating system and it is continuously evolving.

- **Multi-User** - Linux is a multiuser system means multiple users can access system resources like memory/ ram/ application programs at same time.

- **Multiprogramming** - Linux is a multiprogramming system means multiple applications can run at same time.

- **Hierarchical File System** - Linux provides a standard file structure in which system files/ user files are arranged.

Operating Systems Examples (LINUX)

- **Shell** - Linux provides a special interpreter program which can be used to execute commands of the operating system. It can be used to do various types of operations, call application programs.

- **Security** - Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.

System Programming

Lecture 2
Operating Systems Scheduling, Shell, BIOS and Booting Process

Prepared by
Hazim Noman Abed
Hazim_numan@yahoo.com

Operating Systems Scheduling

- Scheduling is divided into various levels.
- These levels are defined by the location of the processes
- A process can be:
  - available to be executed by the processor
  - partially or fully in main memory
  - in secondary memory
  - is not started yet

Operating Systems Scheduling

Types of Scheduling

- Long term scheduling (batch processing): which determines which programs are admitted to the system for execution and when, and which ones should be exited.
- Medium term scheduling (Swapping): which determines when processes are to be suspended and resumed;
- Short term scheduling (CPU Scheduling): which determines which of the ready processes can have CPU resources, and for how long.

Operating Systems and CPUs

- Whenever the CPU becomes idle, it is the job of the CPU Scheduler (a.k.a. the short-term scheduler) to select another process from the ready queue to run next.

- The storage structure for the ready queue and the algorithm used to select the next process are not necessarily a FIFO queue. There are several alternatives to choose from, as well as numerous adjustable parameters for each algorithm, which is the basic subject of this entire topic.

Operating Systems and CPUs

CPU Scheduling

- Select process(es) to run on processor(s)
- Process state is changed from “ready” to “running”
- The component of the OS which does the scheduling is called the scheduler

Operating Systems and CPUs

Basic Concepts

- Almost all programs have some alternating cycle of CPU number crunching and waiting for I/O of some kind. (Even a simple fetch from memory takes a long time relative to CPU speeds.)

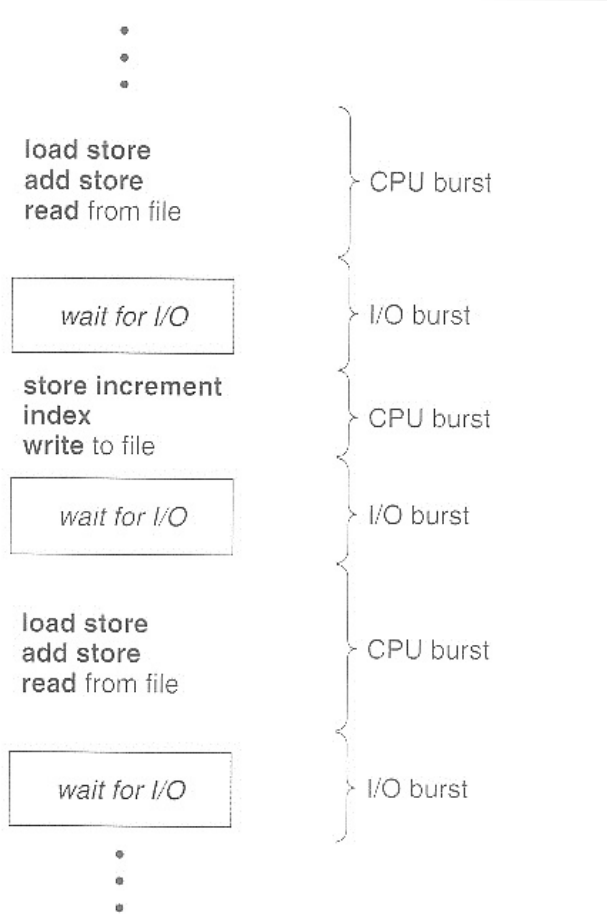
- A scheduling system allows one process to use the CPU while another is waiting for I/O, thereby making full use of otherwise lost CPU cycles.

- The challenge is to make the overall system as "efficient" and "fair" as possible, subject to varying and often dynamic conditions, and where "efficient" and "fair" are somewhat subjective terms, often subject to shifting priority policies.

Operating Systems and CPUs

CPU-I/O Burst Cycle

- Almost all processes alternate between two states in a continuing ***cycle***, as shown in Figure below **:**
  - A CPU burst of performing calculations, and
  - An I/O burst, waiting for data transfer in or out of the system.



Alternating sequence of CPU and I/O bursts.

Operating Systems and CPUs

Preemptive Scheduling

- CPU scheduling decisions take place under one of four conditions:
    1. When a process switches from the running state to the waiting state, such as for an I/O request.
    2. When a process switches from the running state to the ready state, for example in response to an interrupt.
    3. When a process switches from the waiting state to the ready state, say at completion of I/O
    4. When a process terminates.

Operating Systems and CPUs

- For conditions 1 and 4 there is no choice - A new process must be selected.
- For conditions 2 and 3 there is a choice - To either continue running the current process, or select a different one.
- If scheduling takes place only under conditions 1 and 4, the system is said to be non-preemptive, or cooperative. Under these conditions, once a process starts running it keeps running, until it either voluntarily blocks or until it finishes.
- Otherwise the system is said to be preemptive.

Operating Systems and CPUs

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program

- **Dispatch latency** – time it takes for the dispatcher to stop one process and start another running

Operating Systems and CPUs

There are several different criteria to consider when trying to select the "best" scheduling algorithm for a particular situation and environment, including:

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – number of processes that complete their execution per time unit
- **Turnaround time** – Time required for a particular process to complete, from submission time to completion. (Wall clock time.)

- **Waiting time** – How much time processes spend in the ready queue waiting their turn to get on the CPU.

- **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

Operating Systems and CPUs

Scheduling Algorithm Optimization Criteria:

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

Operating Systems and CPUs

First-Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3. The Gantt Chart for the schedule is:

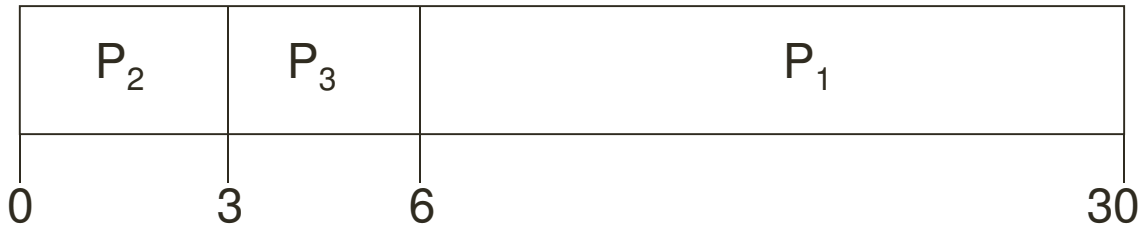
- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

Operating Systems and CPUs

Suppose that the processes arrive in the order:

$$P_2 , P_3 , P_1$$

- The Gantt chart for the schedule is:



- Waiting time for $P_1 = 6; P_2 = 0, P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$
- Much better than previous case

Operating Systems and CPUs

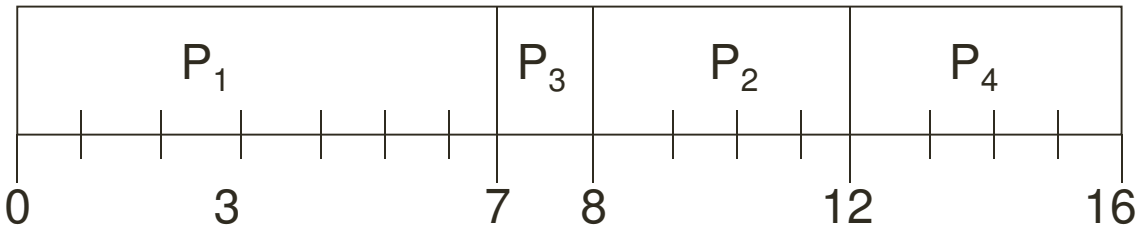
Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request.

Operating Systems and CPUs

Example of Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

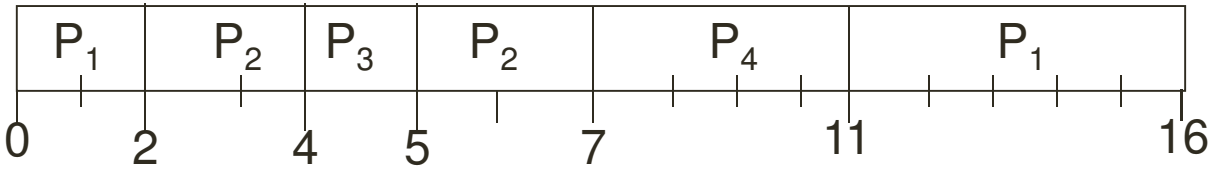


- SJF (non-preemptive)
- Aaverage waiting time = (0 + 6 + 3 + 7)/4 = 4

Operating Systems and CPUs

- Example of Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4



- SJF (preemptive)
- Average waiting time = ((11-2)9 +(5-4) 1 + 0(4-4) +(7-5)2)/4 =3

Operating Systems and CPUs

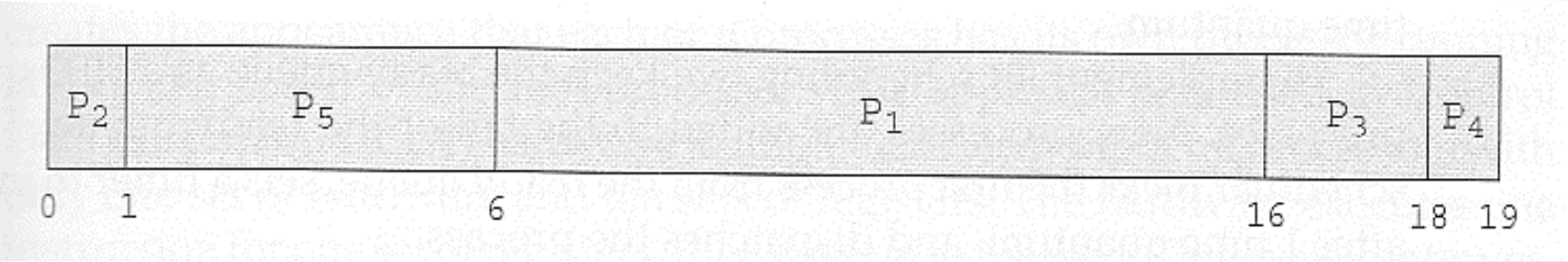
Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - Non-preemptive
- Note that SJF is a priority scheduling where priority is the predicted next CPU burst time
- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process

Operating Systems and CPUs

- **Priority**

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2



Operating Systems and CPUs

Round Robin (RR)

- Implementation:
  - Round robin scheduling is similar to FCFS scheduling, except that CPU bursts are assigned with limits called **time quantum**.

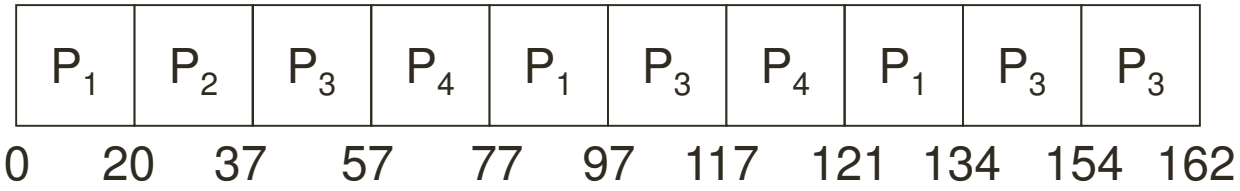
- Characteristics:
  - Preemptive
  - Effective in time sharing environments

Operating Systems and CPUs

- Example of RR with Time Quantum = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- The Gantt chart is:



- Typically, higher average turnaround than SJF, but better *response*.

Shell

- A shell is another term for user interface. Shell in the world of computers refers to a program that allows the user to interact with the computer through some kind of interface.

- The shell is the layer of programming that understands and executes the commands (instructions) which a user enters(writes) in the computer

Shell

- Operating systems provide various services to their users, including file management, process management (running and terminating applications), batch processing, and operating system monitoring and configuration.

- Most operating system shells are not direct interfaces to the underlying kernel, even if a shell communicates with the user via peripheral devices attached to the computer directly.

- A shell manages the user–system interaction by prompting users for input, interpreting their input, and then handling an output from the underlying operating system.

Shell

- Most operating system shells fall into one of two categories – command-line and graphical.
- Command line shells provide a command-line interface (CLI) to the operating system, while graphical shells provide a graphical user interface (GUI).
- Other possibilities, although not so common, include voice user interface and various implementations of a text-based user interface (TUI) that are not CLI.

Shell

Text (CLI) shells

- A command-line interface (CLI) is an operating system shell that uses alphanumeric characters typed on a keyboard to provide instructions and data to the operating system.

- For example, a teletypewriter can send codes representing keystrokes to a command interpreter program running on the computer; the command interpreter parses the sequence of keystrokes and responds with an error message if it cannot recognize the sequence of characters,

- Or it may carry out some other program action such as loading an application program, listing files, logging in a user and many others.

Shell

- Operating systems such as UNIX have a large variety of shell programs with different commands, syntax and capabilities.

- Some operating systems had only a single style of command interface; such as MS-DOS operating system came with a standard command.

- A feature of many command-line shells is the ability to save sequences of commands for re-use. A data file can contain sequences of commands which the CLI can be made to follow as if typed in by a user.

Shell

- Special features in the CLI may apply when it is carrying out these stored instructions. Such as batch files (script files) can be used repeatedly to automate routine operations such as initializing a set of programs when a system is restarted.

- The command-line shell may offer features such as command-line completion, where the interpreter expands commands based on a few characters input by the user.

- A command-line interpreter may offer a history function, so that the user can recall earlier commands issued to the system and repeat them, possibly with some editing.

Shell

Graphical shells

- Graphical shells provide means for manipulating programs based on graphical user interface (GUI), by allowing for operations such as opening, closing, moving and resizing windows, as well as switching focus between windows.

Shell

- Most graphical user interfaces develop the "electronic desktop"(which is a set of unifying concepts used by graphical user interfaces to help users more easily interact with the computer),

- Where data files are represented as if they were paper documents on a desk, and application programs similarly have graphical representations instead of being invoked by command names.

BIOS and Booting Process

BIOS

- The BIOS, short for BASIC INPUT OUTPUT SYSTEM is a set of built-in software routines that give a PC its personality.

- It also manages data flow between the computer's operating system and attached devices such as the hard disk, video adapter, keyboard, mouse and printer.

- System BIOS is a chip located on all motherboards that contain instructions and setup for how your system should boot and how it operates.

BIOS and Booting Process

BIOS and Booting Process

The four main functions of a PC BIOS

- **POST -** The computer power-on self-test (POST) tests the computer to make sure it meets the necessary system requirements and that all hardware is working properly before starting the remainder of the boot process.

- If the computer passes the POST, the computer gives a single beep (with some computer BIOS manufacturers it may beep twice) as it starts up and will continue to start normally.

- However, if the computer fails the POST, the computer will either not beep at all or will generate a beep code, which tells the user the source of the problem.

- If your computer has an irregular POST or a beep code not mentioned below, follow the POST troubleshooting steps to determine the failing hardware component.

BIOS and Booting Process

- **Bootstrap Loader -** Locate the operating system. If a capable operating system is located, the BIOS will pass control to it.
- **BIOS drivers - a**re pieces of software that allow a computer to initialize and boot up correctly
- **BIOS or CMOS Setup -** Configuration program that allows you to configure hardware settings including system settings such as computer passwords, time, and date.

BIOS and Booting Process

Booting Process

- In order for a computer to successfully **boot**, its **BIOS, operating system** and hardware components must all be working properly; failure of any one of these three elements will likely result in a failed boot sequence.

- When the computer's power is first turned on, the CPU initializes itself, which is triggered by a series of clock ticks generated by the system clock. Part of the CPU's initialization is to look to the system's ROM BIOS for its first instruction in the startup program.

- The ROM BIOS stores the first instruction, which is the instruction to run the power-on self test (POST), in a predetermined memory address.

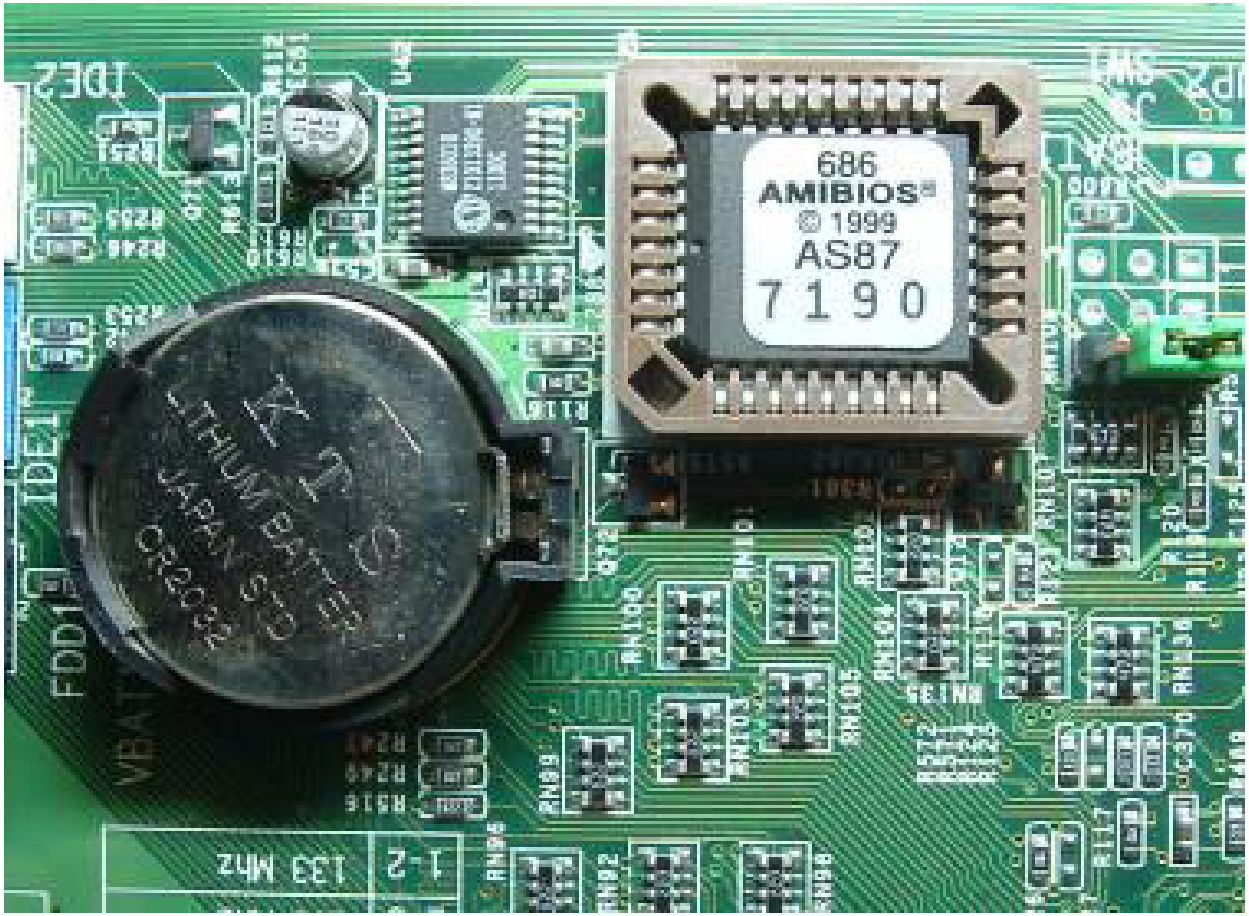
BIOS and Booting Process

- POST begins by checking the BIOS chip and then tests CMOS RAM. (**CMOS** is short for **Complementary Metal-Oxide Semiconductor**. CMOS is an on-board semiconductor chip powered by a CMOS battery inside computers that stores information such as the system time and date and the system hardware settings for your computer)

- If the POST does not detect a battery failure, it then continues to initialize the CPU, checking the hardware devices (such as the video card), secondary storage devices, such as hard drives and floppy drives, ports and other hardware devices, such as the keyboard and mouse, to ensure they are functioning properly.

BIOS and Booting Process

CMOS



Hazim Noman Abed

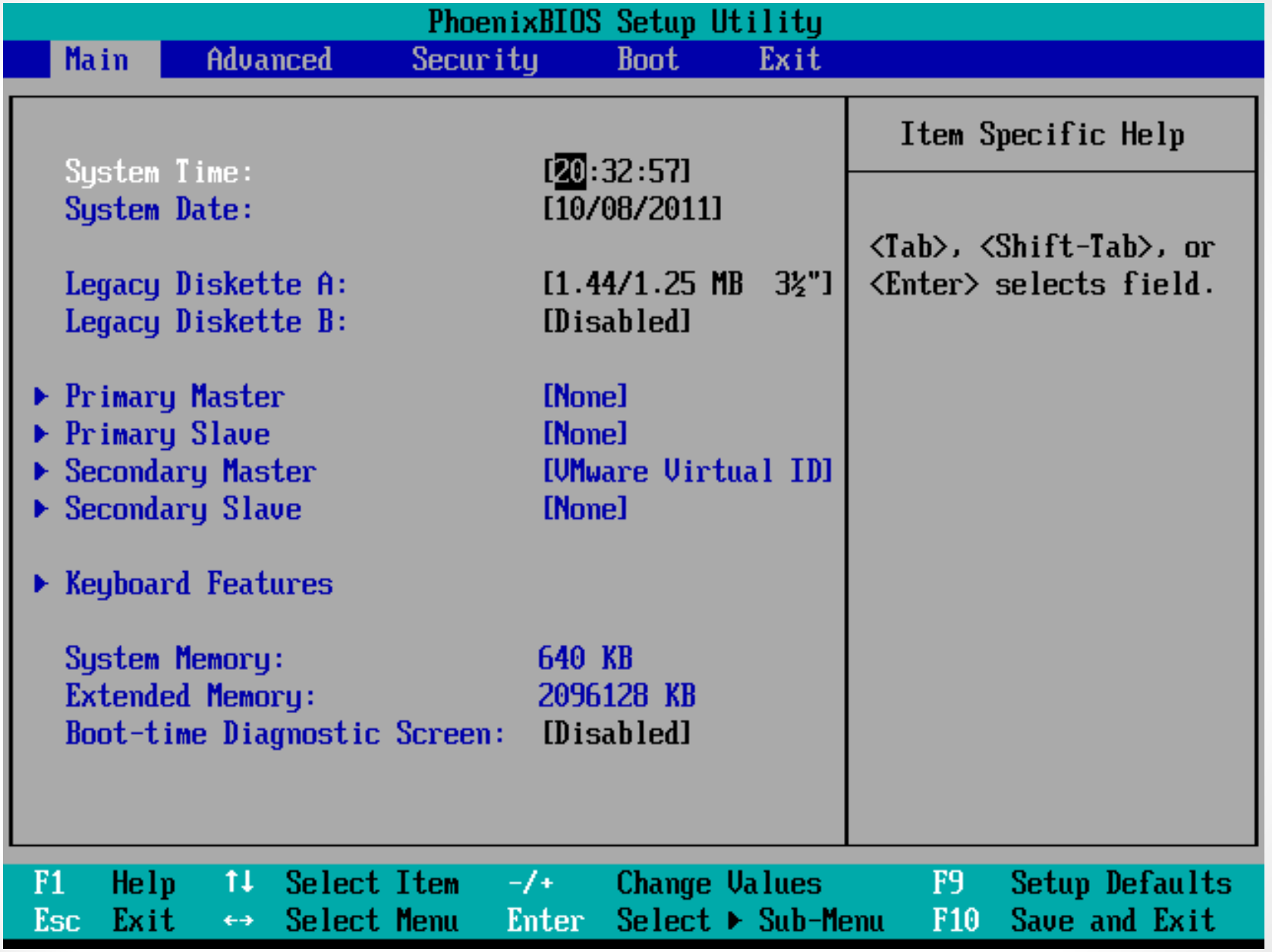
BIOS and Booting Process

- Once the POST has determined that all components are functioning properly and the CPU has successfully initialized, the BIOS looks for an OS to load.

- The BIOS typically looks to the CMOS chip to tell it where to find the OS, and in most PCs, the OS loads from the C drive on the hard drive even though the BIOS has the capability to load the OS from a floppy disk, CD. The process of looks to the CMOS in order to locate the OS is called Boot Sequence

BIOS and Booting Process

- Boot sequence : referred to the boot options and boot order, the boot sequence tells the computer what devices it needs to check and load information from before loading into the operating system. The boot sequence lists the bootable devices in order of priority. The list can be changed by accessing the computer's BIOS, as shown in the example below of the Phoenix BIOS Boot screen.


PhoenixBIOS Setup Utility
Main
Advanced
Security
Boot
Exit
System Time:
[20:32:57]
System Date:
[10/08/2011]
Legacy Diskette A:
[1.44/1.25 MB 3½"]
Legacy Diskette B:
[Disabled]
Primary Master
[None]
Primary Slave
[None]
Secondary Master
[VMware Virtual ID]
Secondary Slave
[None]
Keyboard Features
System Memory:
640 KB
Extended Memory:
2096128 KB
Boot-time Diagnostic Screen:
[Disabled]
Item Specific Help
<Tab>, <Shift-Tab>, or
<Enter> selects field.
F1 Help ↑↓ Select Item -/+ Change Values F9 Setup Defaults
Esc Exit ↔ Select Menu Enter Select ▸ Sub-Menu F10 Save and Exit

BIOS and Booting Process

- Looking to the appropriate boot drive, the BIOS will first encounter the boot sector, which tells it where to find the beginning of the OS and the subsequent program file that will initialize the OS.

- Once the OS initializes, the BIOS copies its files into memory and the OS basically takes over control of the boot process

- Now in control, the OS performs another inventory of the system's memory and memory availability (which the BIOS already checked) and loads the device drivers that it needs to control the peripheral devices, such as a printer, scanner, optical drive, mouse and keyboard. This is the final stage in the boot process, after which the user can access the system’s applications to perform tasks.

System Programming
Lecture 3

Assembler, Compiler, Linker and Loader

Assembler and Compiler

Low level language: Is a programming language that is more arcane and difficult to understand. Some good examples of low-level languages are assembly and machine languages.

High Level Language: Is a languages that is much closer to human language. It's understandable to programmer (Human) and can perform any sort of task, such as C, C++, Pascal.

Assembly language: Is a low level language that deals with hardware registers such as memory addressing and register utilization. Assembly language is in between machine language and high-level language.

- mov al,5 (**Assembly Language**)

Machine Language: Is a collection of binary digits or bits that the computer reads and interprets. Machine language is the only language a computer is capable of understanding.

- 10110000 00000101 (Machine Language)

Assembler and Compiler

What is an Assembler?

- A computer will not understand any program written in a language, other than its machine language. The programs written in other languages must be translated into the machine language.

- Such translation is performed with the help of software. A program which translates an assembly language program into a machine language program is called an assembler.

- If an assembler which runs on a computer and produces the machine codes for the same computer then it is called self assembler or resident assembler.

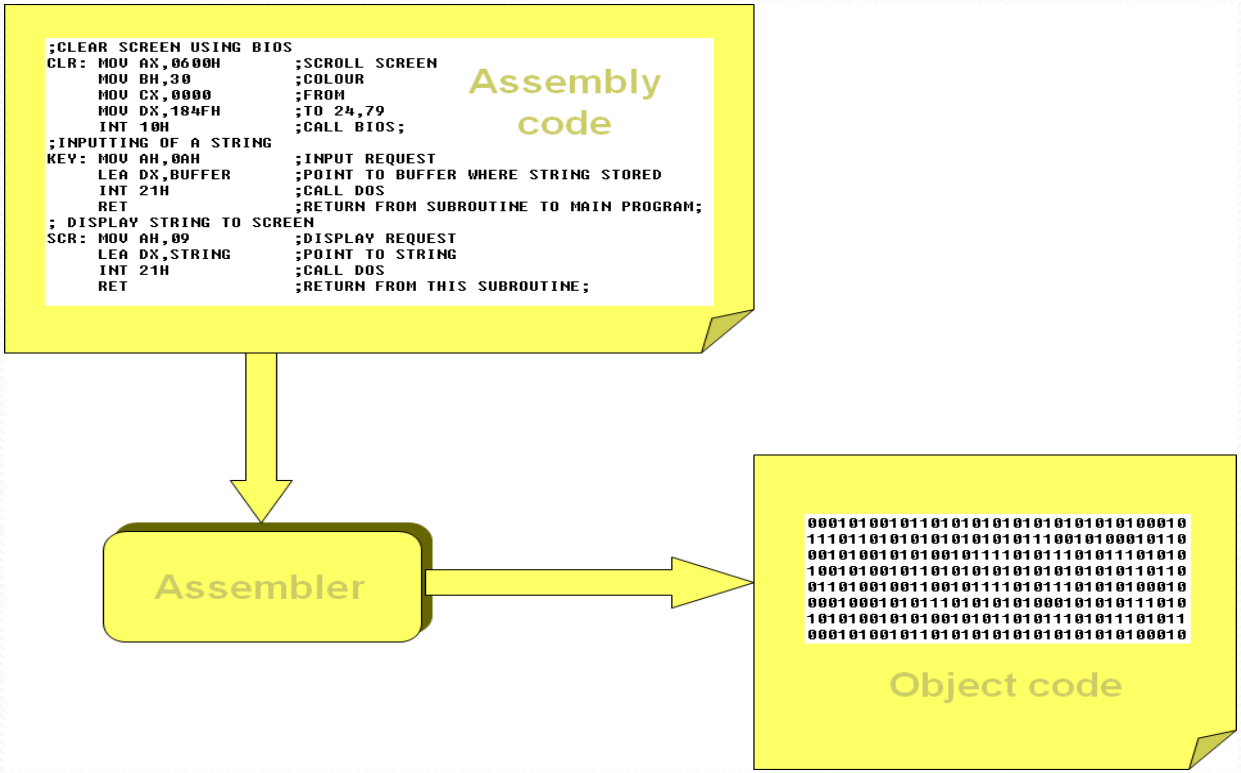
Assembler and Compiler

- If an assembler that runs on a computer and produces the machine codes for other computer then it is called Cross Assembler.

- Assemblers are further divided into two types: One Pass Assembler and Two Pass Assembler.
  1. One pass assembler is the assembler which assigns the memory addresses to the variables and translates the source code into machine code in the first pass as the same time.

Assembler and Compiler

2. A Two Pass Assembler is the assembler which reads the source code twice. In the first pass, it reads all the variables and assigns them memory addresses. In the second pass, it reads the source code and translates the code into object code.



Assembler and Compiler

What is a Compiler?

- Compiler is a computer program that reads a program written in one language, which is called the source language(high level language), and translates it in to another language (low level language), which is called the target language.

- So, in general compilers can be seen as translators that translate from one language to another

- In addition, compilers perform some optimizations to the code. A typical compiler is made up of several main components.

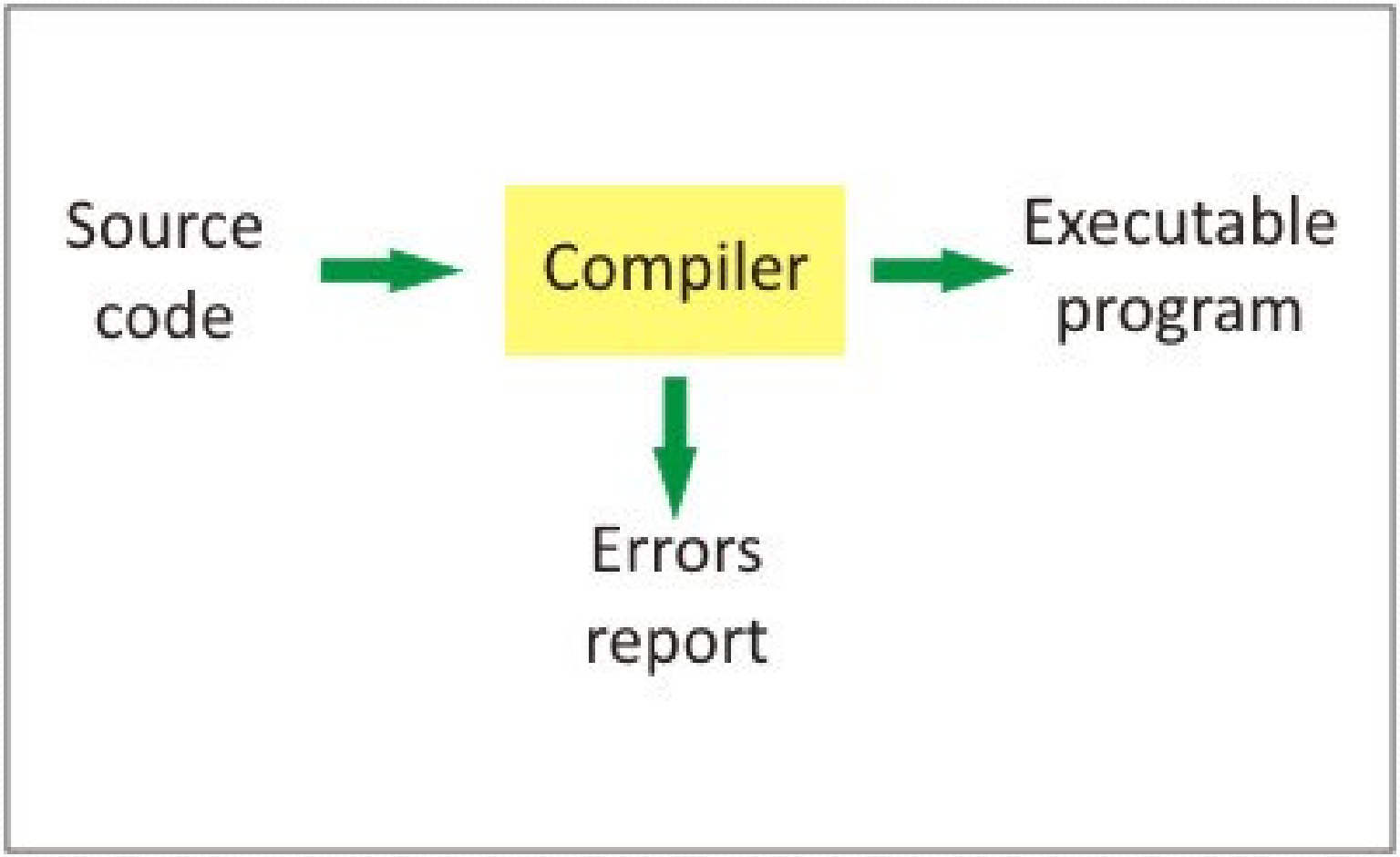
Assembler and Compiler

- The first component is the scanner (also known as the lexical analyzer). Scanner reads the program and converts it to a string of tokens.

- The second component is the parser. It converts the string of tokens in to a parse tree (or an abstract syntax tree), which captures the syntactic structure of the program.

- Next component is the semantic routines that interpret the semantics of the syntactic structure. The code optimizations and final code generation follow this.

Assembler and Compiler



Assembler and Compiler

What is the difference between an Assembler and a Compiler?

- Compiler is a computer program that reads a program written in one language and translates it in to another language,

- while an assembler can be considered a special type of compiler which translates only Assembly language to machine code.

- Compilers usually produce the machine executable code directly from a high level language, but assemblers produce an object code which might have to be linked using linker programs in order to run on a machine.

Assembler and Compiler

- Because Assembly language has a one to one mapping with machine code, an assembler may be used for producing code that runs very efficiently for occasions in which performance is very important (for e.g. graphics engines, embedded systems with limited hardware resources compared to a personal computer like microwaves, washing machines, etc.).

Linker and Loader

What is a Linker?

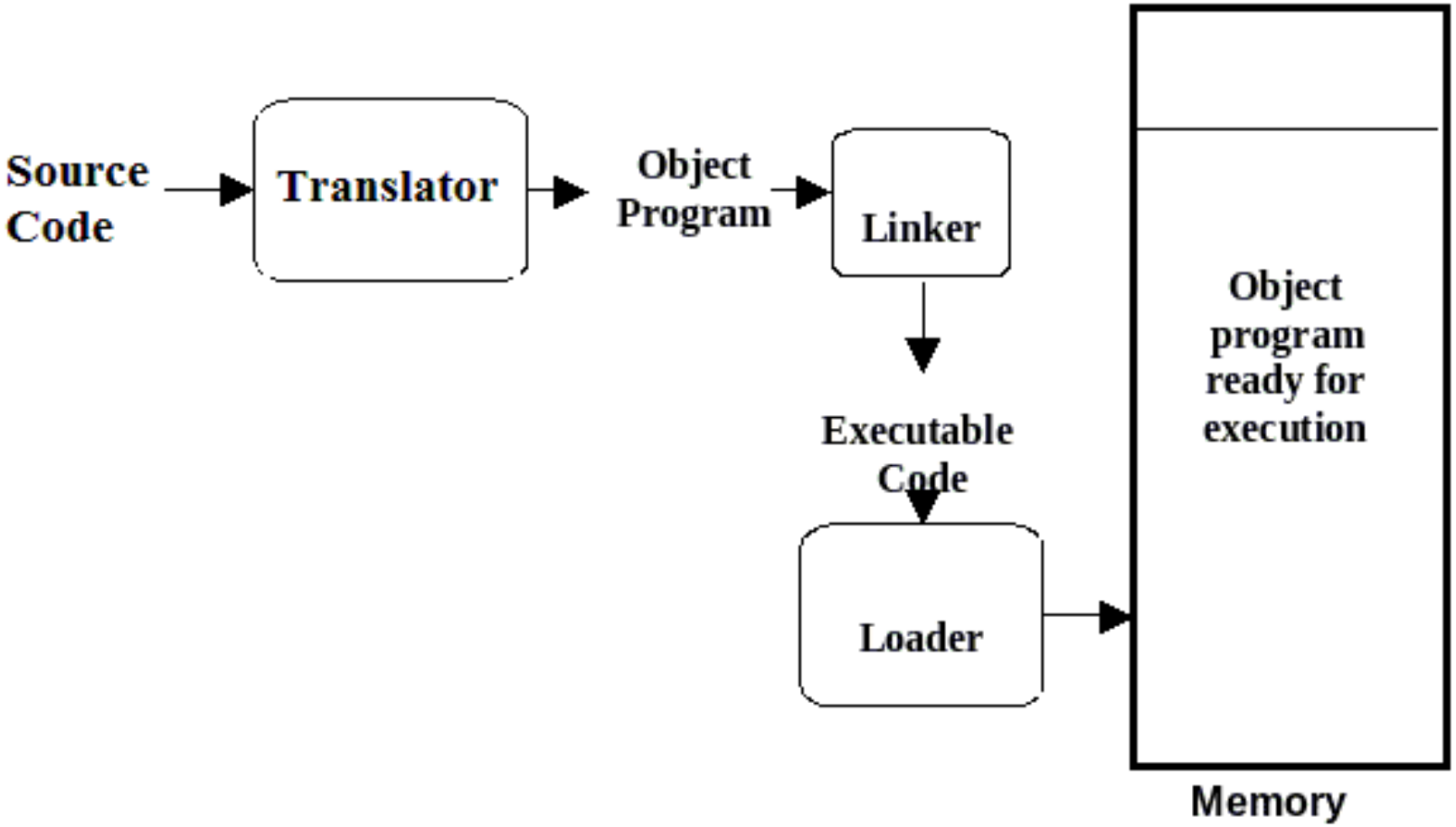
- In high level languages, some built in header files or libraries are stored. These libraries are predefined and these contain basic functions which are essential for executing the program. These functions are linked to the libraries by a program called Linker.

- If linker does not find a library of a function then it informs to compiler and then compiler generates an error. The compiler automatically invokes the linker as the last step in compiling a program.

Linker and Loader

- Not built in libraries, it also links the user defined functions to the user defined libraries. Usually a longer program is divided into smaller subprograms called modules. And these modules must be combined to execute the program. The process of combining the modules is done by the linker.

Linker and Loader



Linker and Loader

What is a Loader?

- Loader is a program that loads machine codes of a program into the system memory.

- In Computing, a **loader** is the part of an Operating System that is responsible for loading programs. It is one of the essential stages in the process of starting a program.

- Because it places programs into memory and prepares them for execution. Loading a program involves reading the contents of executable file into memory.

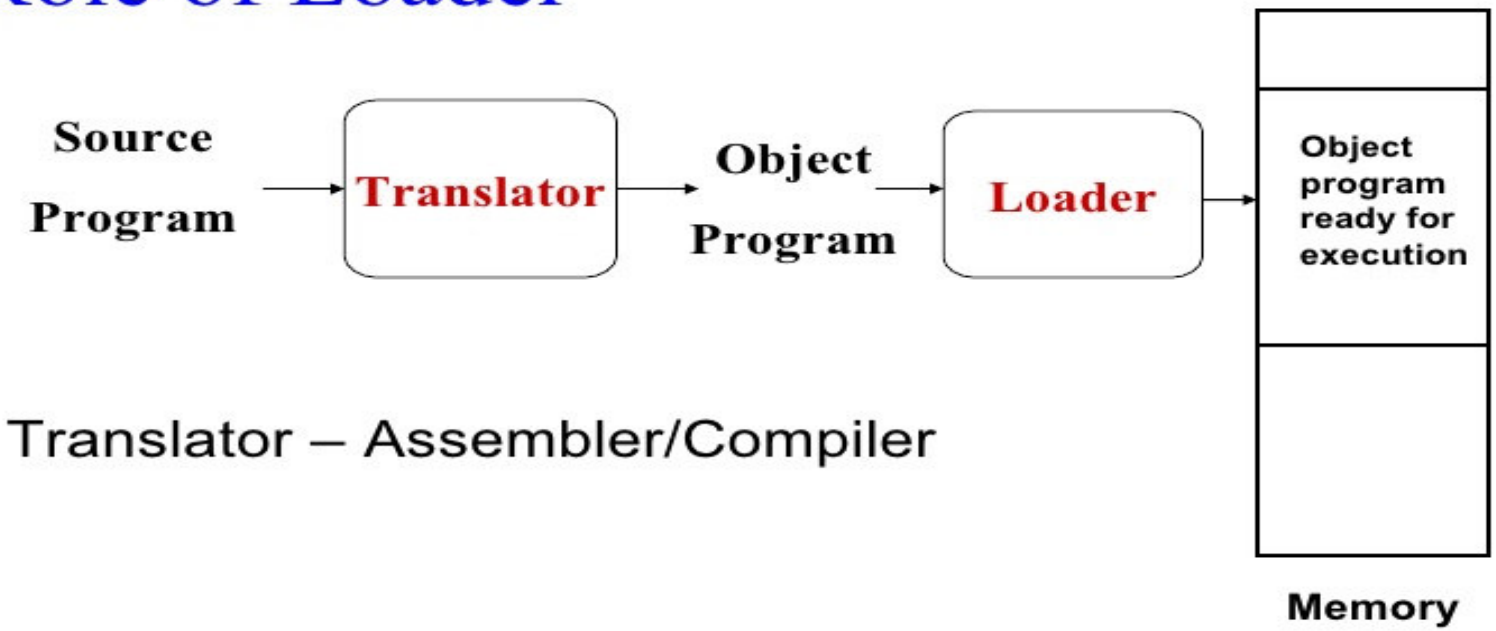
Linker and Loader

- Once loading is complete, the operating system starts the program by passing control to the loaded program code.

- All operating systems that support program loading have loaders. In many operating systems the loader is permanently resident in memory.

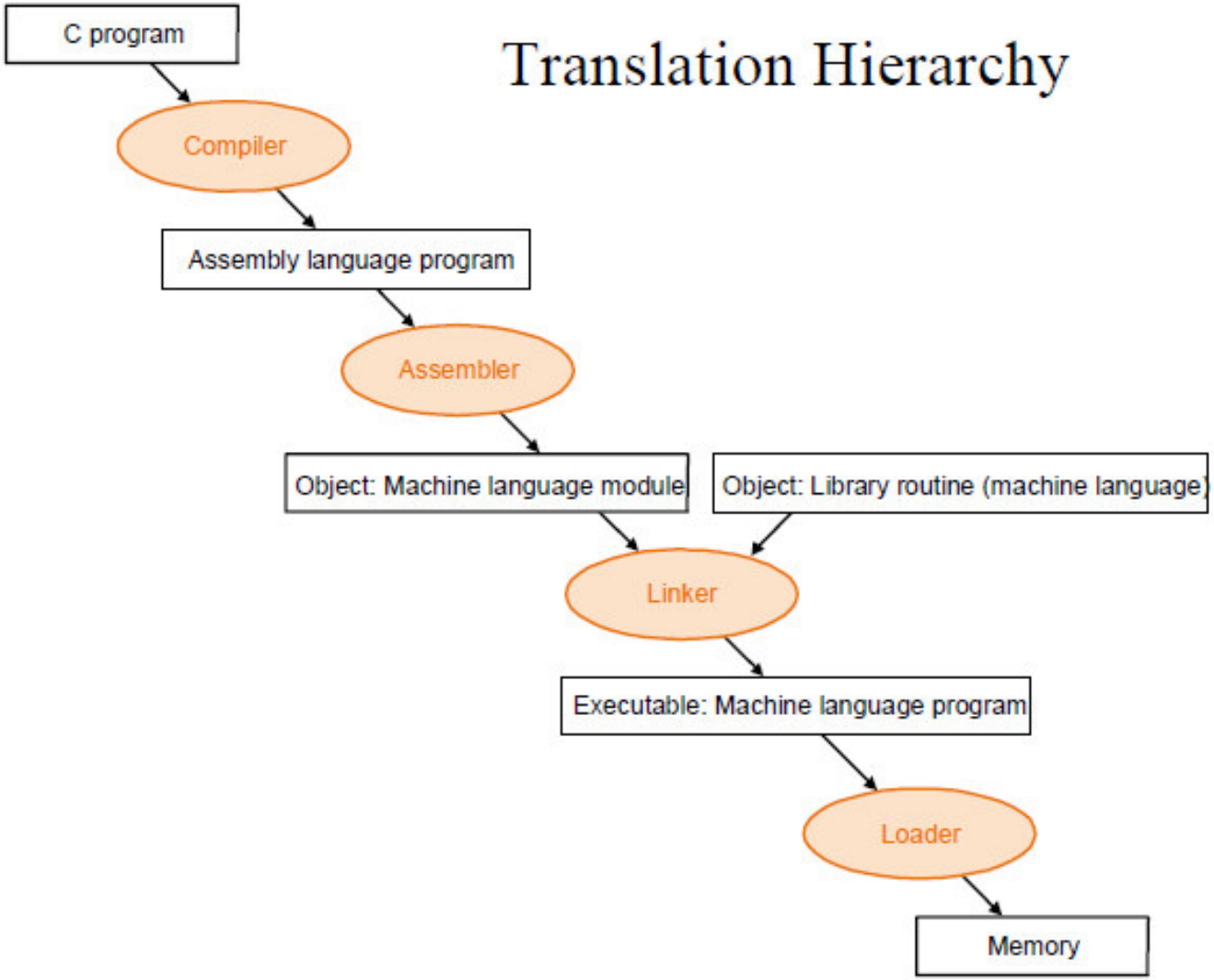
Linker and Loader

Role of Loader



Translator – Assembler/Compiler

Translation Hierarchy



System Programming
Lecture 4
System Calls, API, Interrupt and Exception handling

System Calls

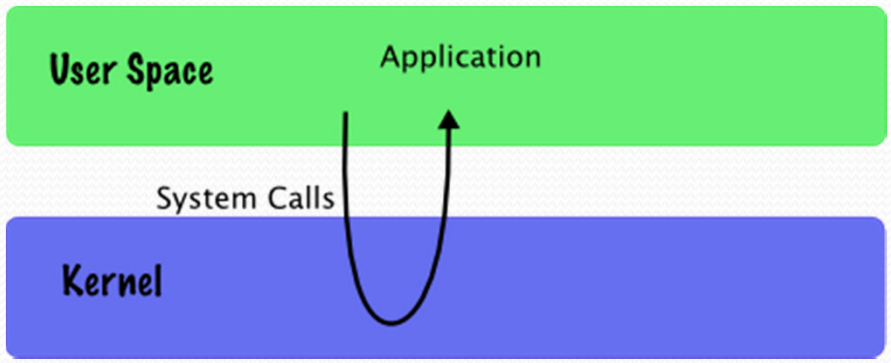
- System calls provide an interface between the process an the operating system. System calls allow user-level processes to request some services from the operating system which process itself is not allowed to do.

- In handling the trap, the operating system will enter in the kernel mode, where it has access to privileged instructions, and can perform the desired service on the behalf of user-level process. It is because of the critical nature of operations that the operating system itself does them every time they are needed

System Calls

- For example, for I/O a process involves a system call telling the operating system to read or write particular area and this request is satisfied by the operating system.

- In computing, a system call is how a program requests a service from an operating system's kernel. This may include hardware related services (e.g. accessing the hard disk), creating and executing new processes, and communicating with integral kernel services (like scheduling). System calls provide an essential interface between a process and the operating system.



System Calls

System calls can be roughly grouped into five major categories:

1. Process Control
   - load
   - execute
   - create process
   - get/set process attributes
   - wait for time, wait event, signal event
   - allocate, free memory

System Calls

2. File management
   - create file, delete file
   - open, close
   - read, write, reposition
   - get/set file attributes

3. Device Management
   - request device, release device
   - read, write, reposition
   - get/set device attributes
   - logically attach or detach devices

System Calls

4. Information Maintenance
   - get/set time or date
   - get/set system data
   - get/set process, file, or device attributes

5. Communication
   - create, delete communication connection
   - send, receive messages
   - transfer status information
   - attach or detach remote devices

API(Application Program Interface)

- Stands for "Application Program Interface," though it is sometimes referred to as an "Application Programming Interface." An API is a set of commands, functions, and protocols which programmers can use when building software for a specific operating system. The API allows programmers to use predefined functions to interact with the operating system, instead of writing them from scratch.

API(Application Program Interface)

- All computer operating systems, such as Windows, Unix, and the Mac OS, provide an application program interface for programmers. APIs are also used by video game consoles and other hardware devices that can run software programs. While the API makes the programmer's job easier, it also benefits the end user, since it ensures all programs using the same API will have a similar user interface.

API(Application Program Interface)

Popular API Examples

1. Google Maps API: Google Maps APIs lets developers embed Google Maps on webpages using a JavaScript or Flash interface. The Google Maps API is designed to work on mobile devices and desktop browsers.

2. YouTube APIs: YouTube API: Google's APIs lets developers integrate YouTube videos and functionality into websites or applications. YouTube APIs include the YouTube Analytics API, YouTube Data API, YouTube Live Streaming API, YouTube Player APIs and others.

Interrupt

Interrupt:

- In systems programming, an **interrupt** is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention.

- An interrupt alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing. The processor responds by suspending its current activities, saving its state, and executing a function called an interrupt handler (or an interrupt service routine, ISR) to deal with the event.

- This interruption is temporary, and, after the interrupt handler finishes, the processor resumes normal activities. There are two types of interrupts: hardware interrupts and software interrupts.

Interrupt

- **Hardware interrupts** are used by devices to communicate that they require attention from the operating system. Internally, hardware interrupts are implemented using electronic alerting signals that are sent to the processor from an external device, which is either a part of the computer itself, such as a disk controller, or an external peripheral.

- For example, pressing a key on the keyboard or moving the mouse triggers hardware interrupts that cause the processor to read the keystroke or mouse position. Unlike the software type (described below), hardware interrupts are asynchronous and can occur in the middle of instruction execution, requiring additional care in programming. The act of initiating a hardware interrupt is referred to as an interrupt request (IRQ)..

Interrupt

- A **software interrupt** is caused either by an exceptional condition in the processor itself, or a special instruction in the instruction set which causes an interrupt when it is executed. The former is often called a *trap* or exception and is used for errors or events occurring during program execution that are exceptional enough that they cannot be handled within the program itself.

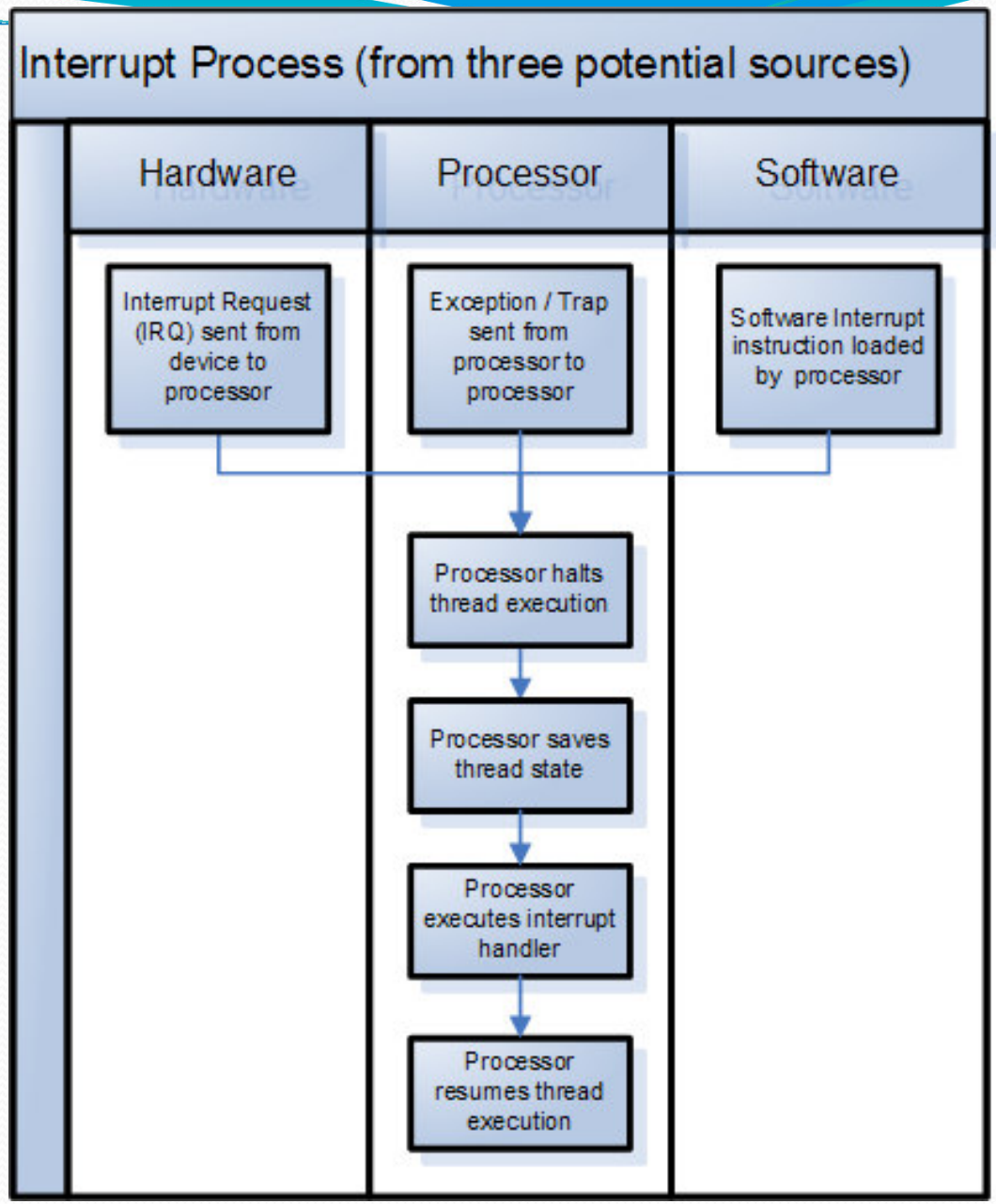
- For example, if the processor's arithmetic logic unit is commanded to divide a number by zero, this impossible demand will cause a *divide-by-zero exception*, perhaps causing the computer to abandon the calculation or display an error message. Software interrupt instructions function similarly to subroutine calls and are used for a variety of purposes, such as to request services from low-level system software such as device drivers.

Interrupt

- For example, computers often use software interrupt instructions to communicate with the disk controller to request data be read or written to the disk.

- Each interrupt has its own interrupt handler. The number of hardware interrupts is limited by the number of interrupt request (IRQ) lines to the processor, but there may be hundreds of different software interrupts. Interrupts are a commonly used technique for computer multitasking, especially in real-time computing. Such a system is said to be interrupt-driven.

Interrupt


Interrupt Process (from three potential sources)
Hardware
Processor
Software
Interrupt Request (IRQ) sent from device to processor
Exception / Trap sent from processor to processor
Software Interrupt instruction loaded by processor
Processor halts thread execution
Processor saves thread state
Processor executes interrupt handler
Processor resumes thread execution

Exception handling

Exception handling is the method of building a system to detect and recover from exceptional conditions. Exceptional conditions are any unexpected occurrences that are not accounted for in a system's normal operation. It is difficult to protect a system from the effects of exceptional conditions because, by nature, all unusual occurrences cannot be anticipated when the system is designed.

Some examples of exceptional conditions are incorrect inputs from the user, bit level memory or data corruption, software design defects that cause a system to enter an undefined state, and environmental anomalies. If these exceptional conditions are not properly caught and handled, they can cause an error or failure in the system. Failures due to exceptions are estimated to account for two thirds of system crashes and fifty percent of system security vulnerabilities

Exception handling

Exception handling techniques can be separated into two broad categories:

1. **Programmed exception handling:**
   - Programmed exception handling modules are mechanisms built into software for specific exceptional cases that are known are likely to occur. Since these occurrences are relatively well understood, protection for them can be incorporated into the system. When a program is executing, if one of the exceptional conditions is detected, control is passed from the main process block to the special exception handling block.

Exception handling

- This code will deviate from normal execution to compensate for the exceptional condition and will attempt to mask it to prevent propagating an error condition to higher levels in the software hierarchy.

- If the condition cannot be recovered, the exception handler may call checkpointing recovery code to return the system to a known state before the exception occurrence and retry the operation.

Exception handling

2. Default Exception Handling

- For all the exceptional conditions that are not anticipated by the system designers, default exception handlers must be built. The default handlers may be within the programming language or operating environment itself, transparent to the application developer. They must be a catch-all for any unexpected exceptions, and must also be responsible for containing exceptions due to design defects.
- Exceptional conditions due to design defects are especially dangerous because they will always be present. If you knew about all design defects in a system a priori, they would have been eliminated before building the system. Since we have not yet learned how to design perfect systems, it is important that exception handlers can reduce the impact of design defects as much as possible.

Exception handling

2. Default Exception Handling

- For all the exceptional conditions that are not anticipated by the system designers, default exception handlers must be built. The default handlers may be within the programming language or operating environment itself, transparent to the application developer. They must be a catch-all for any unexpected exceptions, and must also be responsible for containing exceptions due to design defects.
- Exceptional conditions due to design defects are especially dangerous because they will always be present. If you knew about all design defects in a system a priori, they would have been eliminated before building the system. Since we have not yet learned how to design perfect systems, it is important that exception handlers can reduce the impact of design defects as much as possible.

Exception handling

- In most cases, default exception handlers cannot do much to continue system operation. In the best cases they can use the checkpointing and recovery system to mask transient errors, but for truly exceptional conditions that cause error states, the best that can be hoped for is a graceful program termination.

- In order to achieve robust operation, as much exception handling as possible is desired. However, exception handling overhead may be too great for real-time systems and make timing and scheduling difficult.