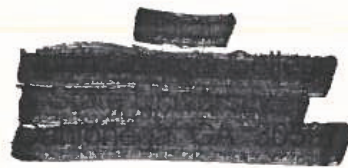


Operating System

Concepts and Techniques

M. Naghibzadeh

Professor of Computer Engineering



iUniverse, Inc.

New York Lincoln Shanghai

**Operating System
Concepts and Techniques**

Copyright © 2005 by Mahmoud Naghibzadeh

All rights reserved. No part of this book may be used or reproduced by any means, graphic, electronic, or mechanical, including photocopying, recording, taping or by any information storage retrieval system without the written permission of the publisher except in the case of brief quotations embodied in critical articles and reviews.

iUniverse books may be ordered through booksellers or by contacting:

iUniverse
2021 Pine Lake Road, Suite 100
Lincoln, NE 68512
www.iuniverse.com
1-800-Authors (1-800-288-4677)

ISBN-13: 978-0-595-37597-4 (pbk)
ISBN-13: 978-0-595-81992-8 (ebk)
ISBN-10: 0-595-37597-9 (pbk)
ISBN-10: 0-595-81992-3 (ebk)

Printed in the United States of America

Contents

Preface	xiii
Chapter 1 Computer Hardware Organization	1
1.1 The Fetch-Execute Cycle	2
1.2 Computer Hardware Organization	4
1.3 Summary	7
1.4 Problems	8
Recommended References	9
Chapter 2 The BIOS and Boot Process	10
2.1 BIOS Actions	11
2.1.1 Power-On Self Test	11
2.1.2 BIOS Manufacturer's Logo	11
2.2 The Operating System Boot Process	13
2.3 Protection Mechanisms	17
2.4 Think of Making Your Own Operating System	20
2.5 Summary	21
2.6 Problems	22
Recommended References	22
Chapter 3 Multiprogramming and Multithreading	24
3.1 Process States in Single-Programming	25
3.2 The Multiprogramming Concept	26

3.3 Process State Transition Diagram	29
3.4 Process Switching	31
3.5 The Interrupt System	32
3.6 Direct Memory Access	35
3.7 Multitasking	36
3.8 Multithreading	37
3.9 Multiprocessing	38
3.10 Summary	39
3.11 Problems	39
Recommended References	40
Chapter 4 Process	41
4.1 Process Attributes	42
4.2 System Calls	47
4.3 Process States and State Transition Model	51
4.4 Process Creation and Termination	61
4.4.1 Why is a Processes Created?	62
4.4.2 When is a Process Created?	62
4.4.3 What is done to Create a Process?	63
4.4.4 What are the Tools for Creating a Process?	63
4.4.5 What is Common in Parent and Child Processes?	65
4.4.6 What is Different Between Parent and Child Processes?	65
4.4.7 How is a Process Terminated?	66
4.5 Process-Based Operating Systems	66
4.6 Summary	67
4.7 Problems	67
Recommended References	68

Chapter 5 Thread	69
5.1 Thread Types	71
5.2 Common Settings for Sibling Threads	73
5.3 Thread Attributes	74
5.4 Thread States and the State Transition Model	77
5.5 Thread Creation and Termination	79
5.5.1 Why Create Threads?	79
5.5.2 When is a Thread Created?	80
5.5.3 How is a Thread Create?	80
5.5.4 What are the Tools for Creating a Thread?	81
5.5.5 How is a Thread terminated?	83
5.5.6 What is done After Thread Termination?	84
5.6 Thread-Based Operating Systems	84
5.7 Summary	84
5.8 Problems	85
Recommended References	85
Chapter 6 Scheduling	86
6.1 Scheduling Criteria	87
6.1.1 Request Time	87
6.1.2 Processing Time	87
6.1.3 Priority	87
6.1.4 Deadline	88
6.1.5 Wait Time	88
6.1.6 Preemptability	88
6.2 Scheduling Objectives	89
6.2.1 Throughput	89

6.2.2 Deadline	89
6.2.3 Turnaround Time	90
6.2.4 Response Time	91
6.2.5 Priority	91
6.2.6 Processor Utilization	91
6.2.7 System Balance	91
6.2.8 Fairness	92
6.3 Scheduling levels	92
6.3.1 High-Level Scheduling	92
6.3.2 Medium-level scheduling	93
6.3.3. Low-level scheduling	94
6.3.4 Input/output scheduling	94
6.4 Scheduling Algorithms: Single-Processor Systems	96
6.4.1 First-Come-First-Served	99
6.4.2 Shortest Job Next	101
6.4.3 Shortest Remaining Time Next	102
6.4.4 Highest Response Ratio Next	102
6.4.5 Fair-Share Scheduling	103
6.4.6 Round Robin	104
6.4.7 Priority scheduling	106
6.4.8 Process Scheduling in Linux	106
6.5 Analysis of Scheduling Algorithms	108
6.5.1 Average Response Time	111
6.6 Multiprocessor Scheduling	113
6.6.1 Processor Types	113
6.6.2 Processor Affinity	113

6.6.2 Synchronization Frequency	114
6.6.3 Static Assignment versus Dynamic Assignment	115
6.7 Scheduling Algorithms: Multi-Processor Systems	116
6.7.1 First-Come-First-Served	116
6.7.2 Shortest Job Next	116
6.7.3 Shortest Remaining Time Next	116
6.7.4 Fair-Share Scheduling	117
6.7.5 Round Robin	117
6.7.6. Gang Scheduling	117
6.7.7 SMP Process Scheduling in Linux	118
6-8 Scheduling Real-Time Processes	118
6.8.1 Rate-Monotonic Policy	119
6.8.2 Earliest Deadline First Policy	121
6.8.3 Least Laxity First Policy	122
6.7 Disk I/O Scheduling	123
6.9.1 First-In-Fist-Out	126
6.9.2 Last-In-Fist-Out	126
6.9.3 Shortest Service Time First	127
6.9.4 Scan	128
6.9.5 Circular-Scan	129
6.10 Summary	129
6.11 Problems	130
Recommended References	132
Chapter 7 Memory Management	133
7.1 Older Memory Management Policies	135
7.2 Non-virtual Memory Management Policies	141

7.2.1 Segmentation Memory Management	141
7.2.2 Relocatable Partition Memory Management	143
7.2.3 Page Memory Management	144
7.3 Virtual Memory	149
7.4 Page-Based Virtual Memory Management	150
7.4.1 Address Translation	155
7.4.2 Page Removal Algorithms	158
7.4.4 Multilevel Page-Table Page-Based Virtual Memory	172
7.5 Segmentation with Paged Virtual Memory	172
7.6 Cache Memory	174
7.6 Windows Two-Level Page-Table Structure	178
7.7 The UNIX Memory Management Policy	181
7.8 Summary	182
7.9 Problems	183
Recommended References	188
Chapter 8 Interprocess Communication/Synchronization	189
8.1 Why Process Communication/Synchronization?	189
8.1.1 Race Condition	190
8.1.2 Critical Region	193
8.1.3 Mutual Exclusion	194
8.2 IPC and Synchronization Techniques	195
8.2.1 Disabling Interrupt Method	196
8.2.2 Busy-Wait-Based IPC	197
8.2.3 Semaphore-Based IPC	204
8.2.4 Monitor-Based IPC	209
8.2.5 Message-Based IPC	211

8.3 Summary	214
8.4 Problems	215
Recommended References	220
Chapter 9 Deadlock and Starvation	221
9.1 Deadlock	221
9.1.1 Deadlock Conditions and Modeling	222
9.1.2 Ignoring Deadlock	224
9.1.3 Deadlock Prevention	225
9.1.4 Deadlock Avoidance	228
9.1.5 Deadlock Detection and Recovery	234
9.2 Starvation	239
9.3 A Classic Example of IPC	239
9.4 Summary	245
9.5 Problems	245
Recommended References	247
Chapter 10 Operating System Categories	248
10.1 Software Technology and Operating Systems	248
10.1.1 Monolithic Operating Systems	248
10.1.2 Structured Operating Systems	249
10.1.3 Object-Oriented Operating Systems	250
10.1.4 Agent-Oriented Operating Systems	250
10.2 Acting Entities and Operating Systems	251
10.2.1 Process-Based Operating Systems	251
10.2.2 Thread-Based Operating Systems	252
10.2.3 Agent-Based Operating Systems	253
10.3 Kernel Types of Operating Systems	254

xii Operating System

10.3.1 Macro-kernel Operating Systems	254
10.3.2 Microkernel Operating Systems	255
10.3.3 Extensible Kernel Operating Systems	256
10.4 Hardware Structures and Operating Systems	256
10.4.1 Single-Processor Operating Systems	257
10.4.2 Multiprocessor Operating Systems	257
10.4.3 Distributed Operating Systems	258
10.4.4 Internet Operating Systems	259
10.5 Timeliness and Operating Systems	260
10.5.1 Real-Time Operating Systems	260
10.5.2 Non-Real-Time Operating Systems	261
10.6 Summary	262
Recommended References	262
References	265
Index	271

Preface

Computers are widely employed in almost all the activities of today's modern man. They are used by people of all ages in a variety of environments. We recognize the computer as a set of devices, such as the main case, monitor, keyboard, mouse, etc., collectively called the hardware. In one session's work with the computer, we turn it on, wait a few seconds until it is ready for use, and then start asking the operating system to run our program of choice. Therefore, it is correct to assume that every computer has an operating system.

The Operating System (OS) is the most essential program of all without which it becomes absolutely cumbersome to work with a computer. It is the interface between the hardware and us (i.e., computer user), making it much easier and more pleasant to use the hardware. Familiarizing ourselves with the operating system and its services helps us to utilize the computer in a more effective manner and to do our jobs well. With this in mind, we need to know the operating system's services and how to make use of them. The operating system itself has to be written so that it can exploit the hardware in the most efficient way possible. Much effort has been put into this matter. Since the computer was invented in the 1940s, many scientists have strived to make this software user-friendly and efficient for the operation of the hardware. The net result is truly a masterpiece that utilizes a multitude of innovations and the fruit of much hard work.

There are tens of operating systems of different sizes and brands, and hundreds of books instructing us how to use them. Also, many innovators and authors have written books on explaining the design concepts of the operating system. The expected readers of the latter are operating system professionals and university students that are taking courses about similar subjects.

In this book, we first discuss a less clarified aspect of the operating system, namely the way the operating system takes over the hardware and becomes the real internal owner of the computer. This knowledge, plus a macro view of what the operating system does and how it does it, provides an insight and

advantage to make better use of the overall facilities of the computer. It also provides the basis for a more comprehensive understanding of the complex aspects of the operating system. Then, we elaborate on concepts, methods, and techniques that are employed in the design of operating systems.

This book is for everyone who is using a computer but is still not at ease with the way the operating system manages programs and available resources in order to perform requests correctly and speedily. High school and university students will benefit the most as they are the ones who turn to computers for all sorts of activities, including email, Internet, chat, education, programming, research, playing games etc. Students would like to know more about the program that helps out (in the background) to making everything flow naturally. It is especially beneficial for university students of Information Technology, Computer Science and Computer Engineering. In my opinion, this book must definitely be read by students taking an operating systems course. It gives a clear description on how advanced concepts were first invented and how they are now implemented in modern operating systems. Compared to other university text books on similar subjects, this book is downsized by eliminating lengthy discussions on subjects that only have historical value.

In Chapter One, a simplified model of computer hardware is presented. We refrained from including any detail that is not directly related to our goals. The fetch-execute cycle of performing instructions is described in this chapter.

In Chapter Two, the role of Basic Input Output System (BIOS) in initial system checking and the operating system boot process is explained. The layered view of the operating system is elaborated on. The kernel (the essential part of the seed of the operating system) and its role in helping fulfill the needs of other layers of the operating system and computer users is talked about. Another important issue called computer security and protection is explained.

Chapter Three describes the reasons behind multiprogramming, multitasking, multiprocessing, and multithreading. The most fundamental requirements for facilitating these techniques are also discussed in this chapter. A brief discussion on process states, the state transition diagram, and direct memory access is provided. The basic foundations of multithreading methodology and its similarities and dissimilarities to multiprogramming are highlighted.

Chapter Four is about processes. Important questions are answered in this chapter, such as: What is a process? Why and how is a process created? What information do we need to keep for the management of every process? How is a process terminated? What is the relationship between a child process and its parent process? When is a process terminated? A short discussion is included on the properties of process-based operating systems. The real-life scenarios of this chapter are based on the UNIX operating system which is a process-based

operating system. This chapter emphasizes the process as the only living entity of process-based operating systems. Possible state transitions of processes are discussed and the role of the operating system in each transition is explained.

Chapter Five focuses on threads. Despite the existence of processes, the reason for using threads is explained along with its relation to processes. Some operating systems are thread-based. Kernel functions of this kind of operating system are performed by a set of threads. In addition, a primary thread is generated as soon as a process is created. Other questions are studied in this chapter, like: What is a thread? Why and how is a thread created? What information do we need to manage every thread? How is a thread terminated? What is the relationship between a child thread and its parent? When is a thread terminated? Examples are taken from the Windows Operating System. Possible state transitions of threads are discussed and the role of the operating system in each transition is explained.

Scheduling is the subject of Chapter Six. It is one of the most influential concepts in the design of efficient operating systems. First, related terms: request time, processing time, priority, deadline, wait time, and preemptability are defined here. Next, scheduling objectives are discussed: maximizing throughput, meeting deadlines, reducing average turnaround time, reducing average response time, respecting priorities, maximizing process utilization, balancing systems, and being fair. Four categories of scheduling, namely high level, medium level, low level, and I/O scheduling, are distinguished. Many algorithms used in scheduling processes in single processor environments are investigated. They include: first come first served, shortest job next, shortest remaining time next, highest response ratio next, fair-share, and round robin. Most of these scheduling algorithms are also usable in multiprocessor environments. Gang-scheduling is also examined for multiprocessor systems. Intelligent rate-monotonic and earliest deadline first algorithms are widely utilized algorithms in the scheduling of real-time systems. These schedulers are introduced in this chapter. The case studies presented are based on the Linux operating system.

Chapter Seven investigates memory management policies. It briefly introduces older policies like single contiguous partition, static partition, dynamic partition, and segmentation memory management. Non-virtual memory management policies are discussed next. Relocatable partition memory management and page memory management are also two other subjects covered. The emphasis of the chapter is on virtual memory management policies. Above all is the page-based virtual memory management policy that is followed by almost all modern operating systems. A good portion of this section is devoted to page removal algorithms. The page table of large programs takes

a great deal of memory space. We have presented multi-level page table organization as a method for making the application of virtual memory techniques to page tables possible. Windows uses a two-level page table and this model is investigated in detail, as a case study of the multi-level page table policy. We carefully studied address translation techniques that are an essential part of any page memory management as well as demand-page memory management policies. Both internal and external fragmentations are looked into. At the end of the chapter, a complementary section on cache memory management is featured. Real-life techniques of presented in this chapter are based on the UNIX operating system.

Processes that run concurrently compete for computer resources. Every process likes to grab its resources at the earliest possible time. A mediator has to intervene to resolve conflicts. This mediator is the interprocess communication/synchronization subsystem of the operating system. Race condition, critical region, and mutual exclusion concepts are first defined. It has been found that by guaranteeing mutual exclusion, conflict-free resource sharing becomes possible. Methods to achieve this are categorized into five classes, namely disabling interrupt, busy-wait-based, semaphore-based, monitor-based, and message-based methods. We have covered these approaches to process communication/synchronization in Chapter Eight.

Deadlock is an undesirable side effect of guaranteeing mutual exclusion. This concept is explained and the ways of dealing with deadlock are discussed. Ignoring deadlock, deadlock prevention, deadlock avoidance, and deadlock detection and recovery are four approaches discussed in Chapter Nine. Another less important side effect is starvation. The dining philosophers' dilemma, as a classic example of interprocess communication problem, is defined. It is shown that straightforward solutions may cause deadlock and starvation. We have provided an acceptable deadlock and starvation-free solution to this problem. In this context, the concept of starvation is elucidated.

Chapter Ten is dedicated to operating system categorization and an introduction to new methodologies in operating system design. With the advances in software technology, operating system design and implementation have advanced, too. In this respect, monolithic operating systems, structured operating systems, object-oriented operating systems, and agent-based operating systems are distinguished. Based on the acting entities, operating systems are categorized as process-based, thread-based, and agent-based. The differences of these categories are highlighted in this chapter. The kernel is an important part of every operating system. We have differentiated among macro-kernel, microkernel, and extensible kernels. Extensible kernel design is a new research subject and much effort is going into producing

commercial extensible kernels. One more categorization parameter is the hardware structure on which an operating system sits. The platform could be single-processor, multiprocessor, distributed system, or Internet. Properties of the corresponding operating systems are described. Real-time systems require special operating systems in order to provide the quality of service necessary for these environments. Real-time and non-real-time operating systems are the last topics that are investigated in this chapter.

I would like to express my gratitude to my wife, son, and daughter for their encouragement and patience. Their support made the writing of this book a pleasant task for me. In addition, I would like to thank Victoria Balderrama for her assistance in proofreading the text.

M. Naghibzadeh

Chapter 1

Computer Hardware Organization

A computer is composed of a set of hardware modules properly connected together to interact in such a way that the total system is reliable and performs its duty in a correct manner. The operating system sits on top of the computer hardware. It uses hardware facilities to do what it is supposed to do. To understand the operating system, it is very helpful if a simplified view of the hardware is shown first. Going into the details of modules and circuits is not necessary and doing so will divert our attention from our main purposes.

A computer is a device that runs programs. A **program** is a set of instructions that is prepared to do a specific job. Different programs are written to do different jobs. This makes a computer able to do different things for us. A program is normally stored on a disk. This may be a Compact Disc (CD), diskette, etc., but it has to be in the computer's Main Memory (MM) in order to run. The main memory of a computer is usually called Random Access Memory (RAM)-see side textbox. As soon as a program is ready for running it is no longer called program, rather, it is called a **process**. We will not be choosy, in this chapter, and however will use program and process interchangeably.

Main memory is called *random access* because the access time of all locations is the same. It does not matter whether the location in the memory is at the beginning, at the end, or anywhere in between.

What brings the program to the main memory? Of course, another program called loader, with the help of the Operating System (OS). Using a computer language, programs are written by human programmers. These programs have to be translated to a computer friendly language that is usually called **machine language**. What performs the translation? Once again, a program called **translator**, with the help of the operating system, does this.

1.1 The Fetch-Execute Cycle

The computer reads an instruction from the RAM and moves it to a temporary place called the **Instruction Register (IR)** in the **Central Processing Unit (CPU)**. It then finds out what has to be done with respect to this instruction and performs the action. The computer repeats this cycle of moving an instruction from the main memory to the CPU and executing it over and over again. A more detail course of action is shown in Figure 1.1.

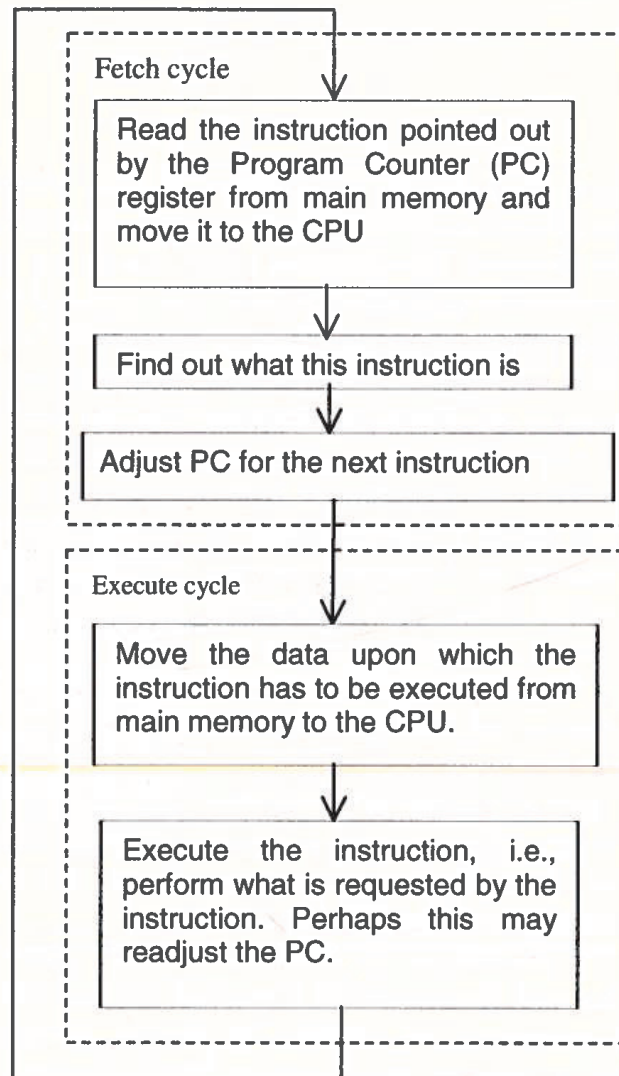


Figure 1.1: A simplified functional model of a computer

In Figure 1.1, the PC is the register that keeps track of what instruction has to be executed next. The PC is automatically filled with an initial value when the computer is started and thereafter it is updated as shown in the figure. Instructions come from a program that is currently being executed. Within a program, when an instruction is executed, the next instruction to be *fetched* is most likely the one immediately following. However, this is not always the case. Look at the piece of program below that is written in pseudo code format.

1. *If* $A > B$
2. *then print* A
3. *else print* B
4. *endif*

Here, we would like the computer to either execute instruction 2 or instruction 3, but not both. There must be a way to jump over instruction 2 when we do not want to execute it. This is actually done by a jump instruction. The execution of a jump instruction is to change the content of the PC to the *address* of the next instruction that has to be executed.

The model presented in Figure 1.1 is for a sequential computer, that is, roughly speaking, a computer with one general processing unit called the CPU. There are other computers called multiprocessor computers that are much faster than sequential computers and can run more than one programs simultaneously. Most of the computers that we are using, e.g., personal computers, are of the sequential type.

During the execution of a program, instructions and data are taken from main memory; therefore, it is necessary to have these ready beforehand. This technique is called *stored-program concept* and was first documented by John Von Neumann. According to this concept, programs, data, and results are

A *register* is a temporary small memory within the CPU.

In some texts, the PC is called the *Location counter* (LC), but when distinction is necessary, we will use LC when talking about a pointer that points to the next instruction of the current program and PC when we mean the register within the CPU that holds the current program's LC.

To *fetch* means moving an instruction from the main memory to the CPU and performing some preliminary actions like *adjusting* LC and *decoding* the instruction (finding out what operation it is.)

A *pseudo-code* is an algorithm (or piece of algorithm) expressed in a simplified language similar to that of a programming language, but is not an actual one.

John Von Neumann, Presper Eckert, John Mauchly, and Hermann Goldstine are on the list of scientists who invented the first electronic digital computer.

stored in the same memory, as opposed to separate memories. The innovation of stored-program was a giant step forward in advancing the field of computer science and technology from punched cards to fast reusable memories.

The central processing unit of a computer is composed of the **Arithmetic Logic Unit (ALU)**, the **Control Unit (CU)**, special registers, and temporary memories. The ALU is the collection of all circuitry that performs arithmetic and logic operations like addition, multiplication, comparison, and logical “AND.” The CU is responsible for the interpretation of every instruction, the designation of little steps that have to be taken to carry out the instruction, and the actual activation of circuitries to do each step. We have already used special registers. Two of which are the instruction register and the program counter. There are other registers for holding current instruction’s results, computer status, and so on. There are also one or more sets of registers that are reserved for the program that is running, (i.e., being executed). Data that is needed frequently, or will be needed in the near future, can be kept in these registers. This way, we save time by not sending them back and forth to the main memory. **Cache memory** is also an essential part of current computers. It is a relatively larger temporary memory that can store a good portion of a program’s data and instructions. They are actually copies of the same data and instructions that reside in main memory.

1.2 Computer Hardware Organization

Within every computer there are many hardware devices. The control unit, arithmetic-logic unit, cache memory, and main memory are basic internal devices. Other devices are either installed internally or as external devices. All of these devices collectively form the **Input/Output Unit (IOU)** of the computer. Figure 1.2 illustrates a hardware model of a **general-purpose computer**. Only essential connections are shown in this figure.

A general-purpose computer is able to run different programs. Therefore, it can perform variety of tasks and it is not restricted to doing one (or very few) tasks. On the other hand, we can think of **special-purpose computers**. A special-purpose computer is designed and built to be used in a specific environment and do one or a few specific tasks, efficiently. In this kind of computer, the program that it runs does not change very often. In reality, all modules of a computer are connected via an internal bus.

The bus can be viewed as a three-lane highway, with each lane assigned to a specific use. One lane is for transferring data, another for transferring addresses and the third for controlling signals.

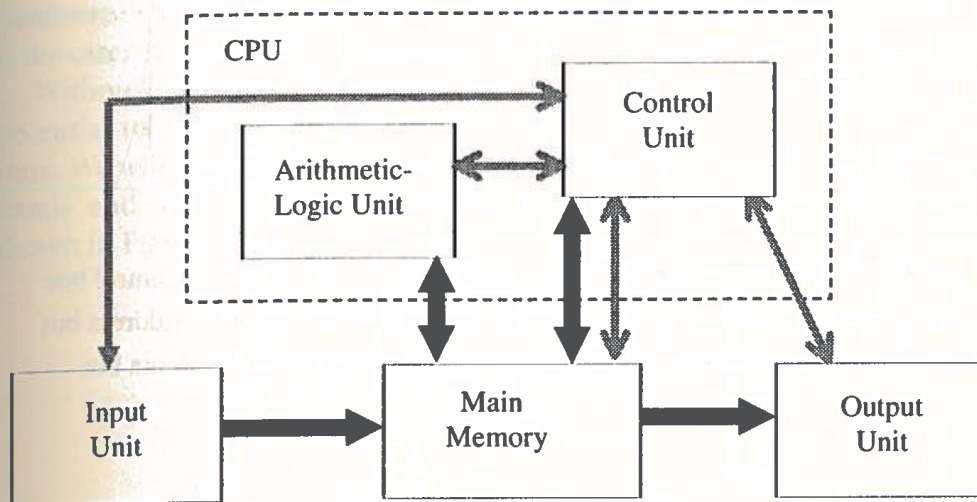


Figure 1.2: Computer organization model

We all know what data is. 123 is a datum, "John" is a datum, etc. If a datum/instruction is needed from the main memory, the requesting device must supply the address in the main memory. The same is true for storing a datum/instruction in the main memory. Addresses travel via the address bus. An address is the identity of a location in the main memory. To access the main memory we must supply the location to be accesses. This is the essential property of contemporary main memories.

Every byte (collection of cells) in the main memory is given an address that facilitates referring to that byte. Addressing starts from zero, which is assigned to the first byte of the memory. By giving consecutive integers to consecutive locations, addressing continues up to the last location of memory.

In a computer every single micro-activity is performed under the control and supervision of the control unit. The control unit commands are sent through the control bus. For its decision-making, the control unit collects information from almost all devices that are connected to the computer. For example, a command should not be sent to a device that is not turned on. The control unit must make sure the device is ready to accept the command. Such information is collected from the device itself.

The majority of computer devices are not directly connected to the computer's internal bus. A device is inserted in a port that is connected to the bus. There are mechanisms for possessing the address bus and data bus, otherwise data and addresses from different devices could become mixed up. A model of the bus-based computer organization is shown in Figure 1.3.

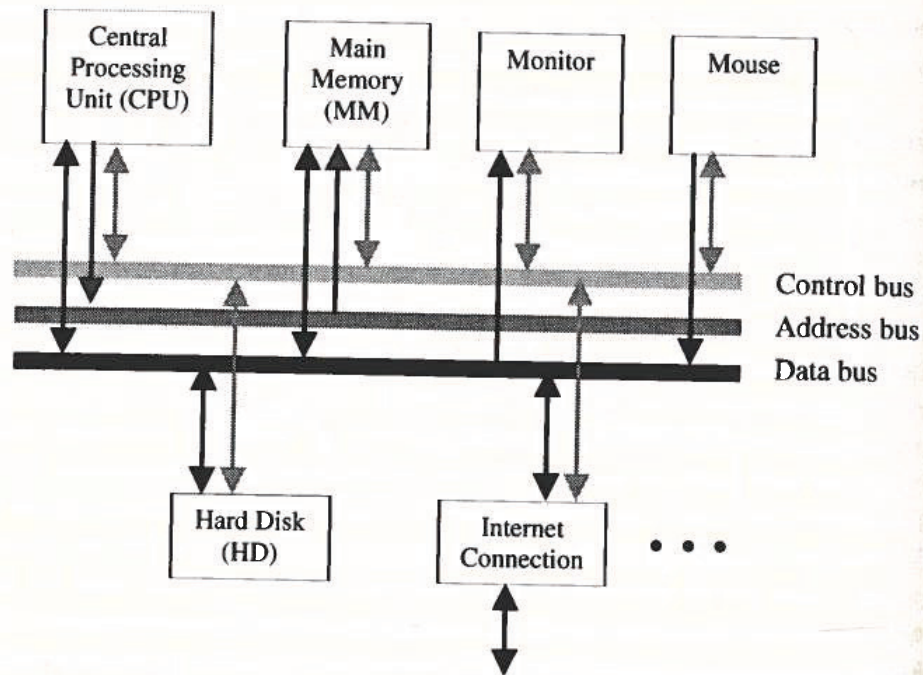


Figure 1.3: Bus-based computer organization

Hardware devices, other than the CPU and MM, have to be installed before use. The mouse, diskette device, light pen, etc. are some of these devices. Installing a device involves registering the device with the operating system and providing a device driver for communication between the device and other parts of the computer. The installation process is started either automatically by the operating system itself or manually by the user. Today's operating systems are intelligent; and once the computer is turned on, the OS recognizes almost all of newly connected hardware devices and immediately begins the installation process. Therefore, a device for which the installation process starts automatically by the operating system and which usually does not require any human intervention, is called a **plug-and-play device**. Few

A *driver* is a piece of software (program) that is used as an interface between a hardware device and the external world. It is usually designed by the device manufacturer in such a way that the device is used in a highly efficient and appropriate way. This is a good reason for using the manufacturer's provided driver. Operating systems have drivers for almost all devices however, due to these driver's generality, they may not work as efficiently and as fast as the manufacturer's specified drivers.

hardware devices have the user manually initiate the installation process. If this is the case, then the OS will interact with the user to complete the process.

Without any software, the computer is called a bare machine, having the potential to perform many functions but no ability to do so in its present state. We will add other features, in a layered fashion, to make our computer handy and user friendly. The current view of our abstracted computer is shown in Figure 1.3.



Figure 1.3: A bare machine

1.3 Summary

From the hardware point of view, a computer is composed of a set of devices, each intended to perform a specific task. The modular design of computer systems has made the task of constructing it much easier and more reliable compared with the monolithic design method. Nevertheless, the design of a computer system is so complex that understanding it requires a certain background including many university courses. In any case, knowledge of all hardware details is not needed to comprehend how an operating system is designed and how it uses hardware facilities to help run programs correctly and efficiently. A simple understanding of computer hardware is absolutely necessary to cover topics in the coming chapters. The fetch-execute cycle explains the microinstructions involved in executing each machine instruction and the many devices and circuits involved in the execution of instructions. This knowledge is immensely helpful for comprehending how programs are executed and how program switching is done. A global organization for computer hardware provides a basis for all kinds of communication among the essential modules of the system.

1.4 Problems

1. The processor checks for an interrupt at the start of each fetch cycle. If there is an unmasked interrupt, the processor has to handle the interrupt(s) first,

before fetching the next instruction. Modify the first part of Figure 1.1 so that it reflects this notion.

2. In the fetch-execute cycle, in order to adjust the PC for the next instruction, the length of current instruction, expressed in bytes, is added to the PC. Categorize your computer's processor instructions based on their length.
3. What are the benefits of adjusting the PC register during the fetch cycle even though we have to readjust it during the execute cycle for some instructions?
4. For a processor that uses two different segments, i.e., the code segment and data segments for instructions and data, respectively, how would you explain the stored-program concept?
5. Explain the advantages and disadvantages of bus-based organization versus point to point connection.
6. Suppose you have just bought a new personal computer. No operating system is installed on it and you are supposed to do this. Is it presently a bare machine? If not, what software does it have and for what purposes?
7. Suppose that your computer is a 32-bit one. Some instructions have no operand and some have one operand field. The operand field is 24 bits long and it can have an immediate value like 2958 or an address of a datum in main memory.
 - a. What is the maximum memory size that your computer can directly address (without using a base register)?
 - b. Now suppose from these 24 bits, one bit is used for a base register. If the content of this bit is zero, the base register is not used. The base register is 32 bits long. An effective address in this case is the contents of the base register plus the contents of the remaining 23 bits of the address field. Now, if the address bus is 32 bits wide, what is the maximum memory size that your computer's processor can directly address?

Recommended References

The concept of stored program was developed by J. von Neumann, J. Presper Eckert, John Mauchly, Arthur Burks, and others following the design of the ENIAC computer [Bur63].

There are many resources about computer organizations and architecture, among which John P. Hayes [Hay03], William Stalling [Sta02], and H. A. Farhat [Far03] have written additional excellent books on the subjects covered in this chapter.