Chapter 6: An Introduction to System Software and Virtual Machines

Invitation to Computer Science, C++ Version, Third Edition
Objectives

In this chapter, you will learn about:

- System software
- Assemblers and assembly language
- Operating systems
Introduction

- Von Neumann computer
  - “Naked machine”
  - Hardware without any helpful user-oriented features
  - Extremely difficult for a human to work with
- An interface between the user and the hardware is needed to make a Von Neumann computer usable
Introduction (continued)

- Tasks of the interface
  - Hide details of the underlying hardware from the user
  - Present information in a way that does not require in-depth knowledge of the internal structure of the system
Introduction (continued)

- Tasks of the interface (continued)

  - Allow easy user access to the available resources
  - Prevent accidental or intentional damage to hardware, programs, and data
System Software: The Virtual Machine

- System software
  - Acts as an intermediary between users and hardware
  - Creates a virtual environment for the user that hides the actual computer architecture

- Virtual machine (or virtual environment)
  - Set of services and resources created by the system software and seen by the user
Figure 6.1
The Role of System Software
Types of System Software

- System software is a collection of many different programs
- Operating system
  - Controls the overall operation of the computer
  - Communicates with the user
  - Determines what the user wants
  - Activates system programs, applications packages, or user programs to carry out user requests
Figure 6.2
Types of System Software
Types of System Software (continued)

- User interface
  - Graphical user interface (GUI) provides graphical control of the capabilities and services of the computer

- Language services
  - Assemblers, compilers, and interpreters
  - Allow you to write programs in a high-level, user-oriented language, and then execute them
Types of System Software (continued)

- Memory managers
  - Allocate and retrieve memory space

- Information managers
  - Handle the organization, storage, and retrieval of information on mass storage devices

- I/O systems
  - Allow the use of different types of input and output devices
Types of System Software (continued)

- **Scheduler**
  - Keeps a list of programs ready to run and selects the one that will execute next

- **Utilities**
  - Collections of library routines that provide services either to user or other system routines
Assemblers and Assembly Language: Assembly Language

- Machine language
  - Uses binary
  - Allows only numeric memory addresses
  - Difficult to change
  - Difficult to create data
Assembly Language (continued)

- Assembly languages
  - Designed to overcome shortcomings of machine languages
  - Create a more productive, user-oriented environment
  - Earlier termed second-generation languages
  - Now viewed as low-level programming languages
Figure 6.3
The Continuum of Programming Languages
Assembly Language (continued)

- Source program
  - An assembly language program

- Object program
  - A machine language program

- Assembler
  - Translates a source program into a corresponding object program
Figure 6.4
The Translation/Loading/Execution Process
Assembly Language (continued)

- Advantages of writing in assembly language rather than machine language
  - Use of symbolic operation codes rather than numeric (binary) ones
  - Use of symbolic memory addresses rather than numeric (binary) ones
  - Pseudo-operations that provide useful user-oriented services such as data generation
.BEGIN  --This must be the first line of the program.

:   --Assembly language instructions like those in Figure 6.5.

HALT  --This instruction terminates execution of the program

:   --Data generation pseudo-ops such as

-.DATA are placed here, after the HALT.

.END  --This must be the last line of the program.

Figure 6.6
Structure of a Typical Assembly Language Program
Examples of Assembly Language Code (continued)

- Algorithmic operations
  
  Set the value of $i$ to 1 (line 2).

  Add 1 to the value of $i$ (line 7).
Examples of Assembly Language Code (continued)

- Assembly language translation

```
LOAD    ONE    --Put a 1 into register R.
STORE   I      --Store the constant 1 into i.

INCREMENT I     --Add 1 to memory location i.

I: .DATA 0     --The index value. Initially it is 0.
ONE: .DATA 1   --The constant 1.
```
Examples of Assembly Language Code (continued)

- Arithmetic expression

A = B + C – 7

(Assume that B and C have already been assigned values)
Examples of Assembly Language Code (continued)

Assembly language translation

```
LOAD   B    --Put the value B into register R.
ADD    C    --R now holds the sum (B + C).
SUBTRACT    SEVEN  --R now holds the expression (B + C - 7).
STORE   A    --Store the result into A.
:         --These data should be placed after the
HALT.
A:  .DATA  0
B:  .DATA  0
C:  .DATA  0
SEVEN:  .DATA  7    --The constant 7.
```
Examples of Assembly Language Code (continued)

- **Problem**

  - Read in a sequence of non-negative numbers, one number at a time, and compute a running sum

  - When you encounter a negative number, print out the sum of the non-negative values and stop
<table>
<thead>
<tr>
<th>Step</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Set the value of Sum to 0</td>
</tr>
<tr>
<td>2</td>
<td>Input the first number $N$</td>
</tr>
<tr>
<td>3</td>
<td>While $N$ is not negative do</td>
</tr>
<tr>
<td>4</td>
<td>Add the value of $N$ to Sum</td>
</tr>
<tr>
<td>5</td>
<td>Input the next data value $N$</td>
</tr>
<tr>
<td>6</td>
<td>End of the loop</td>
</tr>
<tr>
<td>7</td>
<td>Print out Sum</td>
</tr>
<tr>
<td>8</td>
<td>Stop</td>
</tr>
</tbody>
</table>

Figure 6.7
Algorithm to Compute the Sum of Numbers
```
.BEGIN
  CLEAR SUM       --This marks the start of the program.
  IN    N         --Set the running sum to 0 (line 1).
--The next three instructions test whether N is a negative number (line 3).
AGAIN: LOAD ZERO  --Put 0 into register R.
       COMPARE N   --Compare N and 0.
       JUMPLT NEG  --Go to NEG if N < 0.
--We get here if N ≥ 0. We add N to the running sum (line 4).
       LOAD SUM    --Put SUM into R.
       ADD N       --Add N. R now holds (N + SUM).
       STORE SUM   --Put the result back into SUM.
--Get the next input value (line 5).
       IN    N
--Now go back and repeat the loop (line 6).
       JUMP AGAIN
--We get to this section of the program only when we encounter a negative value.
NEG:   OUT SUM    --Print the sum (line 7)
       HALT       --and stop (line 8).
--Here are the data generation pseudo-ops
SUM:   .DATA 0     --The running sum goes here.
N:     .DATA 0     --The input data are placed here.
ZERO:  .DATA 0     --The constant 0.
--Now we mark the end of the entire program.
.END
```

Figure 6.8 Assembly Language Program to Compute the Sum of Nonnegative Numbers
Translation and Loading

Before a source program can be run, an assembler and a loader must be invoked.

Assembler

- Translates a symbolic assembly language program into machine language.

Loader

- Reads instructions from the object file and stores them into memory for execution.
Translation and Loading (continued)

- **Assembler tasks**
  - Convert symbolic op codes to binary
  - Convert symbolic addresses to binary
  - Perform assembler services requested by the pseudo-ops
  - Put translated instructions into a file for future use
Operating Systems

- System commands
  - Carry out services such as translate a program, load a program, run a program
  - Types of system commands
    - Lines of text typed at a terminal
    - Menu items displayed on a screen and selected with a mouse and a button: point-and-click
  - Examined by the operating system
Functions of an Operating System

- Five most important responsibilities of the operating system
  - User interface management
  - Program scheduling and activation
  - Control of access to system and files
  - Efficient resource allocation
  - Deadlock detection and error detection
The User Interface

- Operating system
  - Waits for a user command
  - If command is legal, activates and schedules the appropriate software package

- User interfaces
  - Text-oriented
  - Graphical
Figure 6.15
User Interface Responsibility of the Operating System
System Security And Protection

- The operating system must prevent
  - Non-authorized people from using the computer
    - User names and passwords
  - Legitimate users from accessing data or programs they are not authorized to access
    - Authorization lists
The operating system ensures that

- Multiple tasks of the computer may be underway at one time

- Processor is constantly busy
  - Keeps a “queue” of programs that are ready to run
  - Whenever processor is idle, picks a job from the queue and assigns it to the processor
The Safe Use Of Resources

- **Deadlock**
  - Two processes are each holding a resource the other needs
  - Neither process will ever progress

- The operating system must handle deadlocks
  - Deadlock prevention
  - Deadlock recovery
Historical Overview of Operating Systems Development

- First generation of system software (roughly 1945–1955)
  - No operating systems
  - Assemblers and loaders were almost the only system software provided
Historical Overview of Operating Systems Development (continued)

- Second generation of system software (1955–1965)
  - Batch operating systems
  - Ran collections of input programs one after the other
  - Included a command language
Figure 6.18
Operation of a Batch Computer System
Historical Overview of Operating Systems Development (continued)

- Third-generation operating systems (1965–1985)
  - Multiprogrammed operating systems
  - Permitted multiple user programs to run at once
Fourth-generation operating systems (1985–present)

- Network operating systems

- Virtual environment treats resources physically residing on the computer in the same way as resources available through the computer’s network
Figure 6.22
The Virtual Environment Created by a Network Operating System
The Future

- Operating systems will continue to evolve
- Possible characteristics of fifth-generation systems
  - Multimedia user interfaces
  - Parallel processing systems
  - Completely distributed computing environments
Figure 6.23
Structure of a Distributed System
<table>
<thead>
<tr>
<th>Generation</th>
<th>Approximate Dates</th>
<th>Major Advances</th>
</tr>
</thead>
<tbody>
<tr>
<td>First</td>
<td>1945–1955</td>
<td>No operating system available&lt;br&gt;Programmers operated the machine themselves</td>
</tr>
<tr>
<td>Second</td>
<td>1955–1965</td>
<td>Batch operating systems&lt;br&gt;Improved system utilization&lt;br&gt;Development of the first command language</td>
</tr>
<tr>
<td>Third</td>
<td>1965–1985</td>
<td>Multiprogrammed operating systems&lt;br&gt;Time-sharing operating systems&lt;br&gt;Increasing concern for protecting programs from damage by other programs&lt;br&gt;Creation of privileged instructions and user instructions&lt;br&gt;Interactive use of computers&lt;br&gt;Increasing concern for security and access control&lt;br&gt;First personal computer operating systems</td>
</tr>
<tr>
<td>Fourth</td>
<td>1985–present</td>
<td>Network operating systems&lt;br&gt;Client-server computing&lt;br&gt;Remote access to resources&lt;br&gt;Graphical user interfaces&lt;br&gt;Real-time operating systems&lt;br&gt;Embedded systems</td>
</tr>
<tr>
<td>Fifth</td>
<td>??</td>
<td>Multimedia user interfaces&lt;br&gt;Massively parallel operating systems&lt;br&gt;Distributed computing environments</td>
</tr>
</tbody>
</table>

Figure 6.24
Some of the Major Advances in Operating Systems Development
Summary

- System software acts as an intermediary between the users and the hardware.
- Assembly language creates a more productive, user-oriented environment than machine language.
- An assembler translates an assembly language program into a machine language program.
Summary

- Responsibilities of the operating system
  - User interface management
  - Program scheduling and activation
  - Control of access to system and files
  - Efficient resource allocation
  - Deadlock detection and error detection