

Multimedia Information Networking

Nalin K. Sharda
Victoria University of Technology

Prentice-Hall of India Private Limited

New Delhi - 110 001
2002

Multimedia Information Networking

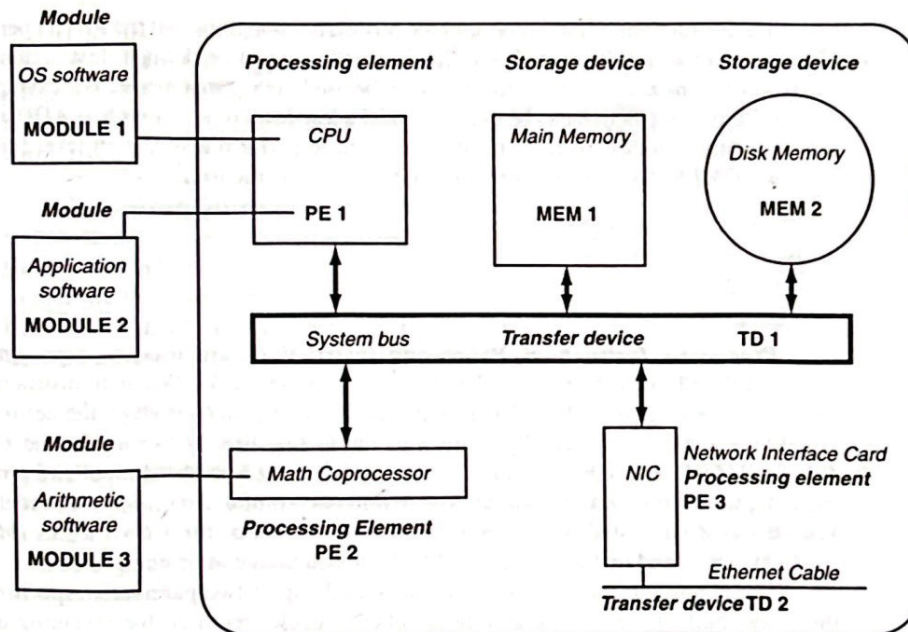
This Indian Reprint—Rs. 250.00
(Original U.S. Edition—Rs. 4231.00)

MULTIMEDIA INFORMATION NETWORKING
by Nalin K. Sharda

© 1999 by Prentice-Hall, Inc. (now known as Pearson Education, Inc.), One Lake Street, Upper Saddle River, New Jersey 07458, U.S.A. All rights reserved. No part of this book may be reproduced in any form, by mimeograph or any other means, without permission in writing from the publisher.

ISBN-81-203-2135-9

Published by Asoke K. Ghosh, Prentice-Hall of India Private Limited, M-97, Connaught Circus, New Delhi-110001 and Printed by V.K. Batra at Pearl Offset Press Private Limited, New Delhi-110015.



Italic = The real-life entity (e.g., CPU, main memory, disk memory)
Bold italic = The term used by NETWORK 11.5 (e.g., processing element)
Bold = A name given to this entity (e.g., PE 1, MEM 2, TD1)

Figure 13.5 Model of a computer connected to a LAN

The user enters a system description by using the NETGEN user interface. This produces a *.NET file. NETWORK is the simulation engine (kernel) that takes the system description from the *.NET file and runs the simulation based on the run-time parameters provided by the user.

Every simulation run produces a number of reports. These reports show utilization, queue statistics, execution time, response time, and conflicts. By analyzing these reports, the user tries to draw conclusions about the system being modeled.

A NETWORK II.5 model consists of hardware and software components. The hardware components are modeled by three types of entities: processing elements, transfer devices, and storage devices. All software components of a system are modeled with the help of modules, instruction mixes, and macro instructions.

13.7.1.1 Processing Elements

A **processing element (PE)** is the basic entity used by NETWORK II.5 to model any device that has the ability to process information. The PE entity can be used to model low-level system components such as a CPU and a bus controller, as well as to model high-level system components such as a database server and a network controller.

Every PE is characterized by the following parameters: *basic cycle time, instruction set, message list, I/O setup time, time slice, interrupt overhead, and input controller.*

The instructions of a PE represent the processing functions that the PE can perform. There is no predetermined level at which instructions must be defined. Instructions that model operations at vastly different levels can be used in the same model. For example, in a given PE one instruction can be used to model a low-level function such as *ADD* and, at the same time, another instruction can be defined to perform a very high-level function such as *COMPRESS IMAGE*. There are four types of instructions:

1. Processing instructions
2. Read/write instructions
3. Message instructions
4. Semaphore instructions

Processing Instructions Processing instructions are used to represent the ability of the PE to perform logical and other processing tasks. When an instruction is executed on a PE in a NETWORK II.5 model, the operation (for which the defined instruction stands) is not actually performed, as in real life. For example, the *COMPRESS IMAGE* instruction will *not* take an uncompressed image as input and produce the compressed image as its output. The main concern of a discrete-event model is to keep track of simulated time. Therefore, the execution of the *COMPRESS IMAGE* instruction will add up the time it would take for an image to be compressed.

The execution time of an instruction is made up of two parameters specified by the model builder: cycle time and the number of cycles required for executing an instruction. For example, if we are modeling an image compression and decompression system, we can define two instructions called *COMP* and *DECOMP*. If these instructions take 100 ms and 105 ms to execute, the basic cycle time can be any value as long as it is a factor of 100 and 105. Two possible options are given in Example 13.4.

Example 13.4 Defining basic cycle time

Problem: Define basic cycle time and number of cycles for *COMP* and *DECOMP* instructions if they take 100 ms and 105 ms, respectively.

Solution:

Option 1: Define the basic cycle time = 1 ms
 Number of cycles for *COMP* = 100
 Number of cycles for *DECOMP* = 105

Option 2: Define the basic cycle time = 5 ms
 Number of cycles for *COMP* = 20
 Number of cycles for *DECOMP* = 21

A statistical distribution function may also be specified as the number of cycles for an instruction. This would be useful to model the behavior of a high-level instruction such as *Perform Join* on the columns of two relations. Since different executions of a join operation can take different lengths of time depending on the size of the table, etc., a fixed value for the execution time is not realistic. Choice of the appropriate statistical distribution function will have to be made based on practical observations or theoretical analysis.

The list of statistical distribution functions available in NETWORK II.5 is quite extensive. These functions are: beta, constant, erlang, exponential, file linear, gamma, IEEE.BACKOFF, key linear, log normal, message linear, normal, pattern, pattern from file, random linear, random step, semaphore linear, triangle, and uniform distribution.

Read/Write Instructions Read/write instructions are used to move files between storage devices and processing elements. A transfer device is used to connect the PE to the SD. Once again, the purpose of simulating such an instruction is to model the time taken by this instruction in performing the data transfer. The total time is a function of the speed of the storage device as well as the speed of the transfer device.

Message Instructions Message instructions are used to transfer data between PEs connected by a TD. On the execution of a message instruction, the message text (which is a part of the instruction) is sent to the destination processor's receive message list. This message stays like a token in the receive message list until it is consumed by a software module. The message text is a message requirement for the module; i.e., this module can execute only if that specific message text is available in the message list.

Semaphore Instructions NETWORK II.5 provides global flags called *semaphores*. Operations on semaphores take zero simulation time. A semaphore instruction has a name, a semaphore, a Set/Reset/Toggle flag, an assignment type, and a count value.

Semaphore status or count can be used by modules to schedule operations. Therefore, semaphores are quite important in implementing the logic of a software system. Binary semaphores are used for implementing synchronization primitives; counting semaphores are useful in modeling buffer management functions.

13.7.1.2 Transfer Device

A **transfer device (TD)** is used to model any channel that can carry bits from one device to another. Any data transfer between PEs and SDs can take place over TDs only. The way in which PEs and SDs are connected determines the topology of the system. The connectivity of a TD is specified as a part of its connection list.

A message instruction must be executed to perform the data transfer over a TD. A message consists of a sequence of blocks. Each block consists of a collection of words, and each word contains one or more bits. Blocks can be used to represent packets on an Ethernet network and frames on a token ring network.

Timing calculation for data transfer over a TD is a function of the six parameters: cycle time, bits per cycle, cycles per word, words per block, word overhead time, and block overhead time.

It is possible to model a variety of packet structures with the list of parameters given above. In Example 13.5, the TD parameters are calculated for transferring 56 bits over a 1,200 baud serial line. Data are organized as 7-bit words with 1-bit overhead per word. Each block comprises 5 words and has a 1 word overhead.