WestVirginiaUniversity
THE RESEARCH REPOSITORY @ WVU

Graduate Theses, Dissertations, and Problem Reports

1998

Real-time multimedia-based education through the Internet

Abhay Arun Bakshi
West Virginia University

Follow this and additional works at: https://researchrepository.wvu.edu/etd

Recommended Citation

Bakshi, Abhay Arun, "Real-time multimedia-based education through the Internet" (1998). *Graduate Theses, Dissertations, and Problem Reports*. 905.
https://researchrepository.wvu.edu/etd/905

This Thesis is protected by copyright and/or related rights. It has been brought to you by the The Research Repository @ WVU with permission from the rights-holder(s). You are free to use this Thesis in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you must obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/ or on the work itself. This Thesis has been accepted for inclusion in WVU Graduate Theses, Dissertations, and Problem Reports collection by an authorized administrator of The Research Repository @ WVU. For more information, please contact researchrepository@mail.wvu.edu.

REAL-TIME MULTIMEDIA BASED EDUCATION THROUGH THE INTERNET

By
Abhay A. Bakshi

A THESIS

Submitted to
The College of Engineering and Mineral Resources
at
West Virginia University

in partial fulfillment of the requirements
for the degree of

Master of Science
in
Electrical Engineering

Department of Computer Science & Electrical Engineering
West Virginia University
Morgantown, WV 26505

Acknowledgements

This research work would have been impossible without the invaluable help and encouragement by my advisor **Prof. Dr. B. J. Das,** working with whom has always been enjoyable for me.

I also particularly want to thank my other committee members, **Dr. Larry Hornak** and **Dr. Wils Cooley** for their priceless suggestions to organize my thesis work in its entirety.

I also thank the most important people in my life: my Mom and Dad and the rest of my family and my friends (especially Vinay Krishna and Kishore Danduboina) who incessantly prodded me to finish my nearing thesis work.

Abhay Bakshi.
September 15, 1998

Table of Contents

List of Figures

Chapter 1 Introduction

The rapid advances in computer and communications technologies, together with the shifting market conditions, are challenging the American education systems to provide increased educational opportunities beyond the traditional geographic boundaries at a reasonable cost. Many educational institutions are answering this challenge by developing distance education programs. At its most basic level, distance education takes place when a teacher and student(s) are separated by physical distance, and technology (i.e., sound, video, data, and print) is used to bridge the instructional gap. Distance education programs can provide adults with a second chance at a college education, reach those disadvantaged by limited time, distance or physical disability, and can provide continuing education to employees at their work places.

A number of different technologies are available for the delivery of distance education; the technologies can be classified as described below [1].

(a) Voice : instructional audio tools include the interactive technologies of telephone, audio conferencing, and short-wave radio. Passive tools include tapes and radio.

(b) Video : instructional video tools include still images such as slides, pre-produced moving images (e.g., film, videotape), and real-time moving images combined with audio-conferencing.

(c) Data : the term ‘data’ is used to describe the broad category of instructional tools involving electronic computers. Computer applications are varied and include:

Computer-assisted instruction (CAI) – uses the computer as a self-contained teaching machine to present individual lessons.

Computer-managed instruction (CMI) – uses the computer to organize instruction and track student records and progress. The instruction itself need not be delivered via a computer, although CAI is often combined with CMI.

Computer-mediated education (CME) – describes computer applications that facilitate the delivery of instruction.

(d) Print : the foundational element of distance education that is the basis from which all other delivery systems have evolved. Typical print formats are: textbooks, study guides, workbooks, course syllabi, and case studies.

Distance education provides effective learning environment. Research comparing distance education to traditional face-to-face instruction indicates that teaching and studying at a distance can be as effective as traditional instruction, when the method and technologies used are appropriate to instructional tasks, there is student-to-student interaction, and when there is timely teacher-to-student feedback. [2] – [3]. An effective means to allow such interactions is through the use of interactive multimedia courseware over the Internet. Interactive multimedia combines the strengths of computer graphics, hypertext, digital video, and audio to provide a superior learning environment.

An important requirement for effective distance education is the availability of continuous interactivity and immediate feedback. Although such capabilities can be provided over the Internet easily for text discussions, synchronous sharing of non-text information (Multimedia content), including video, audio, and common graphics may introduce

unacceptable delays as network bandwidth is limited. Media-rich web pages certainly take a long time to load. For example, fetching an uncompressed 640 X 480 (VGA) image with 24 bits per pixel (922 KB) takes about 4 minutes over a 28.8 kbps modem line. Since sharing, discussing, observing and understanding media-rich document is significant in distance learning, the Internet-bandwidth restriction problem needs to be addressed, which is the objective of this research.

In this thesis, the issue of Internet bandwidth is minimized and a novel system is developed for distance education using multimedia course material over the Internet. The underlying principle of the system is that the data rich multimedia materials reside at remote computers and are only remotely activated utilizing low bandwidth signals over the Internet [4]. As a result, the large download times associated with multimedia files are eliminated creating an almost real time link between the remote locations. The system developed will also allow the remote students to participate in lively class room discussions, and will be an effective means for the delivery of distance education.

Chapter 2 Multimedia Technology and Web Delivery

2.1 Multimedia

Multimedia is the combination of two or more continuous media. These media need to be played during some well-defined time interval with some user interaction. Multimedia uses computer graphics, animation, hypertext, digital video and sound to provide a superior learning environment by stimulating all of the senses. Another strength of multimedia is its interactivity, which allows a self-paced learning environment. When combined with valid contents and sound educational methods, multimedia can provide a superior knowledge attainment and retention. Recent studies by the Department of Defense on the effectiveness of interactive multimedia indicate a reduction in average learning time of 31% and an increase in achievement by 38%. Firms in the *Fortune 1000* report that 16% of today's company training is multimedia-based which is expected to double to 35% within the next two years. Due to its many advantages, multimedia coursewares are finding more and more acceptance in the academic community. Although their numbers are low, multimedia coursewares are now available in every discipline. It is expected that these numbers will increase and that multimedia will play an important role in future education systems.

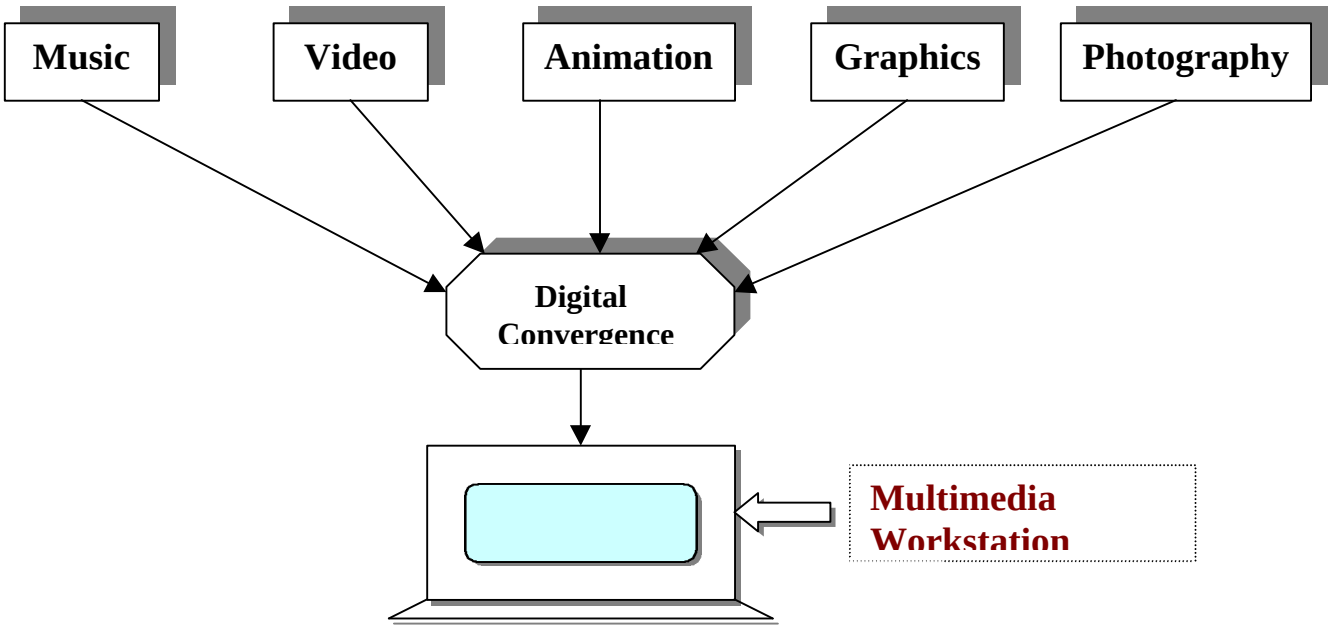


Figure 2.1

A typical multimedia application file is large in size. For delivery of multimedia content to users on a network, there are several available procedures. Among all currently available techniques for multimedia content distribution, the following sections describe the principal methods.

2.2 World Wide Web (WWW)

The WWW is a hypertext information and communication system popularly used on the Internet computer network with data communications operations according to a client-server model. Web clients (browsers) can access multiprotocol and hypermedia information (where helper applications are available for the browsers) using an addressing scheme.

The WWW is an information and communication system. The Web allows both information dissemination and information collecting (through the Forms capability of the

Hypertext Markup Language). Thus, the Web is a two-way system for disseminating information, and includes the potential for interactive communication. Using Forms with gateway programming, Web developers can create systems for user manipulation or change of hypertext structure. As an information dissemination system, the Web can reach audiences of arbitrary size, ranging from just the creator (for hypertext deployed only on a personal file system), or a group (for hypertext deployed on a file system allowing group access), or a mass audience(for hypertext made publicly available on Web servers). Web's development is always marked by rapid commercialization and technical change. The first operating prototype of WWW was released by late 1990. Later, by May 1995, there were more than 15000 known public web servers as against 50 by January 1993 [6]. Web and related resources are expected to have self-sustained rapid growth thereafter.

The WWW uses data communications operating according to a client-server model. The client-server model for networked computer systems involves three components: the client, the server and the network. A client is a software application that runs on the end-user's computer host. A server is a software application that runs on the information provider's computer host. Client software can be customized to the user's hardware system, and acts as an interface from that system to information provided on the server. The user can initiate a request for information or action through the client

software. This request travels over the network to the server. The server interprets the request and takes some desired action. This action might include a database lookup or a change in recorded database information. The results of the requested transaction (if any) are sent back to the client for display to the user. All client-server communication follows a set of rules, or protocols, which are defined for the client-server system.

Figure 2.2 below summarizes these relationships, showing the flow of a request from a client to a server and the transmission of information from a server to a client.

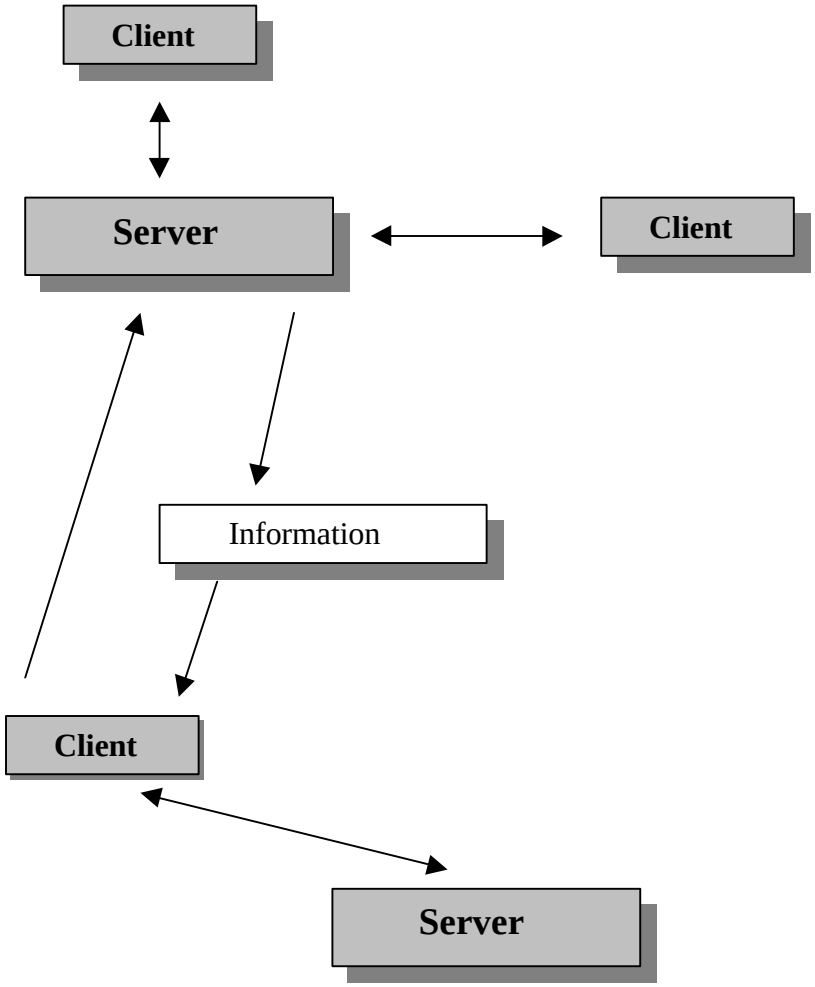


Figure 2.2

A client might access many servers employing the protocol(s) that both the server and client understand. In this research, a centralized model of communication between several clients and a single server is utilized, and will be discussed in more detail in the next chapter.

Web browsers and Multimedia :

Web browsers display multimedia content primarily in three different ways: [7]

1. Native, inline. Native, inline media can be displayed directly on the Web page without any additional programs or viewers. All graphical browsers natively support GIF (Graphics Interchange Format) graphics. Some also natively support JPEG (Joint Photographic Experts Group) graphics and other media types.

2. Helper applications. Helper applications are a way for web browsers to display or process MIME (Multipurpose Internet Mail Extensions) types and multimedia content that the browsers cannot display natively. Multimedia content is downloaded to the client's hard disk and is displayed using a non-native helper application such as MoviePlayer.

3. Inline with external code modules such as Netscape-compatible Plug-Ins or Java Applets.

These external 'mini-programs' enable one to play back multimedia content directly in the Web page. These features appeared in Web browsers beginning with Netscape Navigator 2.0. Several multimedia Plug-Ins are automatically installed with Netscape Navigator 3.0.

2.3 Current Technologies for Multimedia Delivery over the Internet :

2.3.1 Integrating CD-ROM Servers into the World Wide Web

CD-ROM technology has become a vital mass-storage tool for more and more organizations. Their ability to deliver huge amounts of data easily, securely, and cost-effectively also make CD-ROMs critical components for data storage and distribution solutions that support and complement the World Wide Web and the growing number of corporate "intranets" utilizing web technology. Installation of CD-ROM servers on a

corporate company-wide network is one possible way for distributing a wide range of information. CD-ROM servers can also serve for the need of distribution of information on World Wide Web. They provide totally virus-free distribution. They can also be easily managed and updated.

Although CD-ROM technology is excellent for storage of large amount of data, it is not suitable for distance education which requires interactive communication between collaborating machines.

2.3.2 Java Technology

Java, developed by Sun Microsystems, is a promising technology for multimedia delivery over the Internet. Java programming language creates executable content which can be distributed through the networks. This content, once downloaded by the user's browser, is displayed in the browser. Executable content is nothing but byte-codes. A Java program, called applet, after compilation produces bytecodes. A Java-enabled browser with the built-in Java Virtual Machine (JVM) or interpreter recognizes a special hypertext tag called APPLET. When downloading a Web page containing an APPLET tag, the browser knows that an applet is associated with that web page. The browser then downloads another file of information, as named in an attribute of the APPLET tag, that describes the execution of that applet. The browser interprets these bytecodes and runs them as an executable program on the user's host. The resulting execution on the user's host then drives the animation, interaction, or further communication. This execution of content on the user's host is what sets Java content apart from the hypertext and other multimedia content of the Web. Also, since a browser comes with built-in JVM, Java technology

makes the distribution and display of multimedia content platform independent. Some of the ways applets are currently being used on the World Wide Web include the following:

- Animation
- Displaying images with sound
- Graphic effects, such as scrolling text
- Interactive programming, such as games

Limits of Applets

Applets can be quite robust programs; however, there are significant differences between applets and applications. Applets are not full-featured applications. There are concrete and implicit limitations that need to be considered in designing and creating applets.

Applets are precompiled bytecode. The compiled bytecode is downloaded to user's machine and executed locally. Because of this, both Sun and browser manufacturers have placed some restrictions on the functionality of applets.

a. Functional Limits

Applets are not given full access to the local machine's file system. Applets currently do not have the means to save files out to a user's local machine, and they can't read files off the local machine either.

Applets are also restricted from loading dynamic or shared libraries from other programming languages. Java has the capability to declare methods as being *native,* which allows the virtual machine to take advantage of other language libraries such as C or C++. However, because these libraries can't be verified and they would require access to the file system, applets are not allowed to use dynamic libraries.

b. Limitations imposed by the browser

In addition to the limitations imposed by the language itself, the browser that executes applet bytecode also places some restrictions on applets. A Web browser is a trusted application—a user has chosen to load the browser on the local machine and trusts that it will not damage any files or operating system. Applets are untrusted applications. They are downloaded immediately and are executed on a machine without receiving the user's consent. Because of this, Java-capable browsers have placed some restrictions on applets in order to ensure that they don't violate system integrity inadvertently.

The most important limitation placed on an applet by the Web browser is network connectivity. Applets currently are limited to network connections with their host machines. This means your Java applet can communicate directly back to the Web server that it's downloaded from, but it cannot contact any other servers on the Net.

c. Other Limitations

Not all browsers have Java bytecode interpreters, for example, browsers for Windows 3.1 systems. Also Java applets can run slow because they are interpreted on the local machine. Java applets can take a long time to download, the big obstacle for all Web Multimedia. This problem is compounded with some Java applets, because if an applet is gathering pieces of class libraries scattered around the Internet, connections may time out if traffic is high, and the applet will not be able to get the pieces it needs to run.

Poorly written Java applets can consume processor cycles and freeze out other interactivity.

Java has primitive multimedia support. Sound support is particularly lacking, being limited to 8-bit, 8kHz *.au* format of a sound file at this time.

Java applets are the last element to load in a Web page; therefore, even if an applet implements streaming playback, it can take a while for the playback to start.

Java and Multimedia

Java's multimedia capabilities are currently primitive and sometimes inadequate or nonexistent. Nevertheless, Java is positioned as an important technology for Internet-based multimedia, particularly for distributing multimedia content on demand. Furthermore, as bandwidth increases—both within corporate intranets and throughout the Internet—and Java matures and is more widely supported, its importance will grow.

Although it currently supports some multimedia features such as graphics, imaging, basic sound, and media tracking, it is still lacking in many ways. This situation really is to be expected, because the Web community in general has yet to figure out the best approach to take in handling distributed multimedia. There are other technologies aiming to solve many of the multimedia problems facing the Web as well.

2.3.3 Shockwave Technology [8]

What is Shockwave?

Shockwave in itself is not the tool that creates multimedia content but rather it is the means by which that content is delivered to Web browsers. Shockwave is a series of plug-ins that enable the playback of streaming, high-quality content in a Web browser.

Shockwave compromises two components.

a. Shockwave plug-in, available at the time of this writing only for a few browsers,

   Netscape Navigator being the most popular.

b. Afterburner

a. Shockwave plug-in

The shockwave plug-in, once installed, plays multimedia files in a Web browser such as Netscape Navigator or Microsoft Internet Explorer. The HTML for inserting the files into a Web page is a simple addition. The files, however, must be created using Macromedia authoring tools.

b. Afterburner

The other half of Shockwave is a small program called Afterburner.

Afterburner compresses multimedia files. For example, afterburner compresses Director movies at a ratio of greater than 2:1. The compressed Director movie is then ready to be placed on any Web server along with the accompanying HTML documents.

Shockwave is not a single software package that adds multimedia to the Web. It's a conversion process. The multimedia (motion, sound, and so on) is created in another software package, then can be viewed over the Internet using Shockwave as sort of a bridge.

Shockwave for Director [5] is the one common form of multimedia over the Internet, but for complex multimedia projects, Shockwave for Authorware [5] is a good choice. Shockwave for Authorware can use much larger, more detailed multimedia files. A third form of Shockwave is Freehand, a vector graphics program. It allows the user to scale and pan a detailed image over the Web.

Shockwave uses the process of streaming. The way a normal Shockwave movie works is to download the entire movie across the Internet, then play it within the browser. In earlier

versions of Shockwave, all sound files needed to be included within the movie. So if the movie contained a 60K audio clip, the whole thing would need to be loaded before it could be played. Streaming separates the sound file from the rest of the Shockwave movie. Then it breaks the sound file into little pieces that can be loaded a little bit at a time as needed. Streaming allows Shockwave to load only a few seconds at a time for real-time audio. It starts by loading a few seconds as a buffer, then immediately plays those few seconds while more data is loaded.

Shockwave Benefits

Notes from Macromedia's official site: Macromedia's Claims →

With Shockwave one can enjoy the best of the Web, including but not limited to games, animated interfaces, interactive ads and demos, streaming CD-quality audio for music and speech, instructional and educational presentations etc. Moreover, shockwave player is free and easy to get. The Shockwave player is available free from www.macromedia.com. It is included with many popular browsers and services such as Netscape Navigator, Microsoft Internet Explorer, and AOL. Shockwave is compatible with Windows 3.1, Windows 95, Windows NT, Macintosh, and Power Macintosh.

Macromedia claims that in 1997, over 40 million Shockwave players were successfully downloaded from Macromedia's Web site.

Problems and Limitations with Shockwave

- Macromedia's Shockwave is still being developed on a daily basis, so there are still some problems and limitations.

  For instance,

  An immediate problem one may find is that after installing Shockwave and

trying to run the multimedia presentation through Netscape's Navigator, you get an error dialog box stating that it's unable to

locate the external movie driver for the embedded Director files. After hours of testing and such, it may be found out that a custom installation of the Director external movie drivers needs to be done.

- The *Afterburner interface* is in the testing stage. Users demand an interface that clearly describes options and presents many help features.
- As far as limitations go, Shockwave deals with the boundaries of technology in several ways. First of all, there are speed limitations when using a product over a local area network. Designers must keep the size of video and audio elements small because of slow download times.

2.4 The World Wide Wait

The glamour that surrounds the Web and multimedia contributes to the user's expectation of full-screen, full-motion, TV-quality, interactive multimedia streaming off their Web pages. The reality, of course, falls far short of this ideal.

The biggest roadblock to delivering multimedia on the Web is the time it takes to download rich, multimedia content over slow network lines. Although some multimedia display capabilities, such as JPEG compression or video playback from hard disk, are dependent on the speed and processing power of the local computer, the main limiting factor for Multimedia on Web is the long download times of large multimedia files. In most cases, the multimedia element has to be completely downloaded to the user's disk before it can be viewed. New streaming technologies are starting to address this problem

by enabling playback to begin before download is complete; however, the overall download time remains still significant.

A top concern for multimedia content providers is keeping file sizes to a minimum to reduce the download time. This restriction usually means a reduction in media quality, such as fewer colors in graphics or a slower frame rate for video. Due to this, multimedia authors usually have to find a balance between file size and the quality.

The large download time affects both home and institutional users. Majority of home users connect to the Internet via modems ranging from 28.8 kbps-56 kbps (kbps = kilobits per second). Since home users usually pay per minute charges for Internet access, large download times are undesirable. On the other side, corporate and university users usually reside on a shared local area network; thus moving large multimedia files to individual workstations impacts performance of the network for all users on the local network.

Data Rate and Bandwidth :

Data transfer rate or data rate is the amount of information transferred to the user's display per unit of time. It is typically measured in kilobytes per second. Data rates are usually limited by user's hardware. Data rates for multimedia playback from 1x and 2x CD-ROM are typically in the range of 100-250 kilobytes per second.

Internet users refer to a similar concept called **bandwidth**. Bandwidth refers to the amount of data that can be transferred down a wire or connection per unit time. It is typically measured in kilobits per second. Most studies indicate the majority of home users are still connecting to the Internet at 14.4 kilobits per second or less. The bandwidth on a dedicated ISDN (Integrated Services Digital Network) connection ranges from about 64-128 kilobits per second. Users on local area networks typically have bandwidth in the

same general range, dependent on network traffic. On shared local area networks, such as TCP/IP-based intranets, this available bandwidth is divided up between multiple users. The actual bandwidth available to an individual user depends on network traffic. The following table [7] compares the data rate and bandwidth capabilities of typical Web connections with more familiar multimedia delivery systems-CD-ROM and television.

Data Rate versus Bandwidth at Popular Connection Speeds

Connection	Data Rate (KB/s)	Bandwidth(Kbps)
28.8 Modem	1.6	28.8
Single line ISDN	8.2	65.5
T1	192	1536
CD-ROM	150	1200
TV	27000	216000

These figures have several implications for multimedia on the Web:

- Multimedia is largely a “wait for it to download and then play it” experience.
- To achieve data rates comparable to CD-ROM over current modems, streaming technologies have to achieve compression ratios that are 10-100 times they are for CD-ROM.
- Internal networks have the bandwidth necessary for the delivery of many types of multimedia, although they still have less bandwidth than a slow CD-ROM.

Chapter 3 System Proposal, Design and Implementation

3.1 Principle of Operation for Proposed System

As discussed in chapter 2, the biggest impediment to multimedia transmission over the Internet is the large delay associated with downloading of multimedia files. It seems unlikely that this problem will be remedied in the near future. It's expected that the continuous explosion of the web together with the increased information glut will rapidly saturate the increased bandwidth of the next generation Internet. Thus, it is very likely that information rich multimedia files will continue to involve appreciable delay time while transmitted through the Internet.

For some applications such as medical imaging and live video, it will always be necessary to transmit large multimedia data files. However, for applications such as education where the multimedia content is pre-developed, transmission of such large files is not necessary. Instead, the data intensive multimedia content can reside at the remote location and the different program components can be remotely activated by transmitting very small activation data files, thus eliminating the delay in downloading large multimedia files. Such remote activation of multimedia program components is the principle based on which the distance education system was developed in this research, and is illustrated schematically in figure 3.1 below.

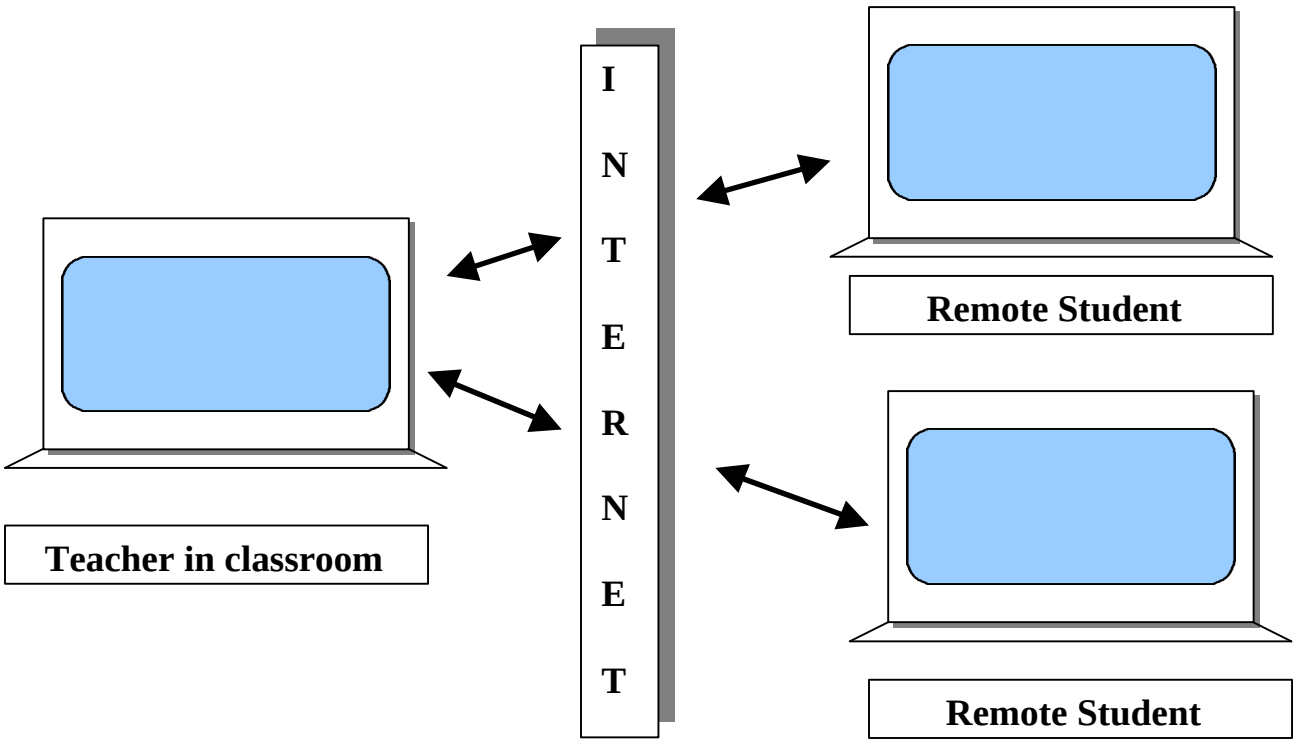


Figure 3.1

In the local or classroom environment, the teacher uses multimedia course material for teaching. Students connected to the Internet from remote locations are active participants in the on-going class session. Multimedia presentation of the courseware runs on the teacher's machine. The same material is seen by the remote students on their computers. Navigation pattern on teacher's machine is synchronously followed on every individual student's machine giving an effect of a close-to real-time virtual classroom. This synchronous operation can be controlled using the teacher's machine acting as a server for all the remote machines as clients.

3.2 The Client Server Model of Interaction

The scene described in section 3.1 is identical to the established client-server protocol on a TCP/IP network, shown in the figure 3.2 below.

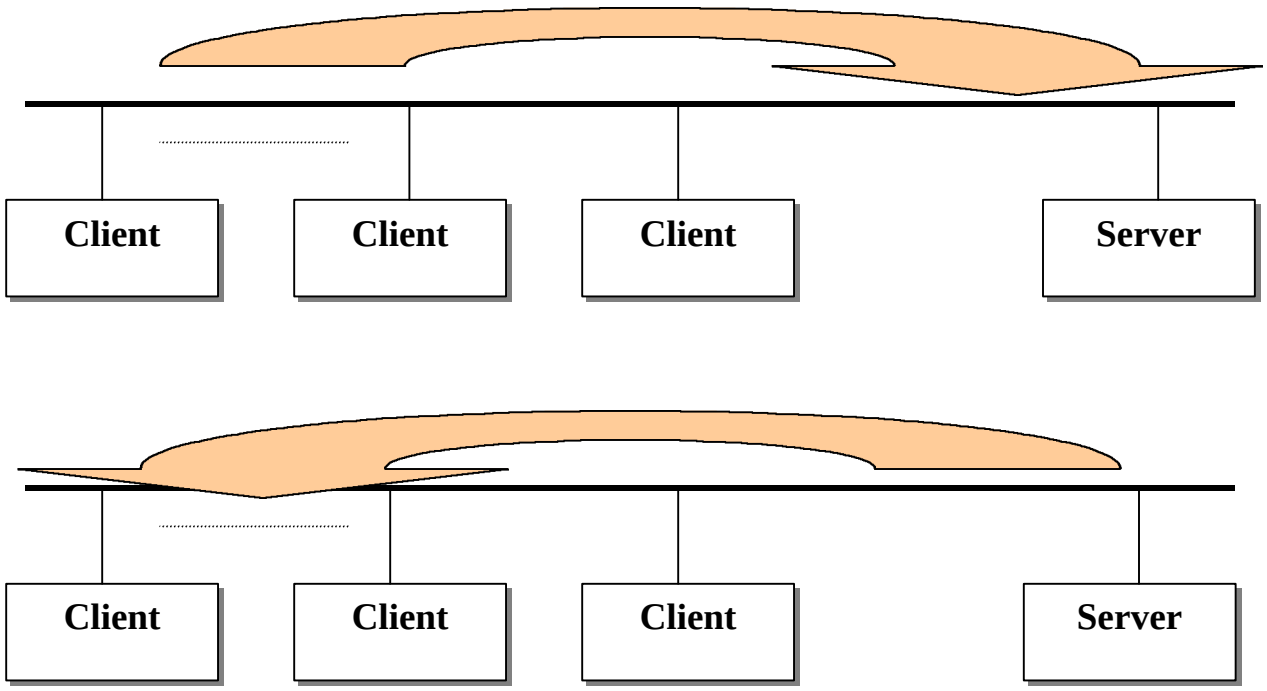


Figure 3.2

As can be seen from the above figure, there are ‘n’ number of clients connected to a server. All the clients (student machines, in our case) are synchronously responded by the server (teacher’s machine, in our case).

3.3 System Implementation

Figure 3.3 below explains the system blueprint and model of execution.

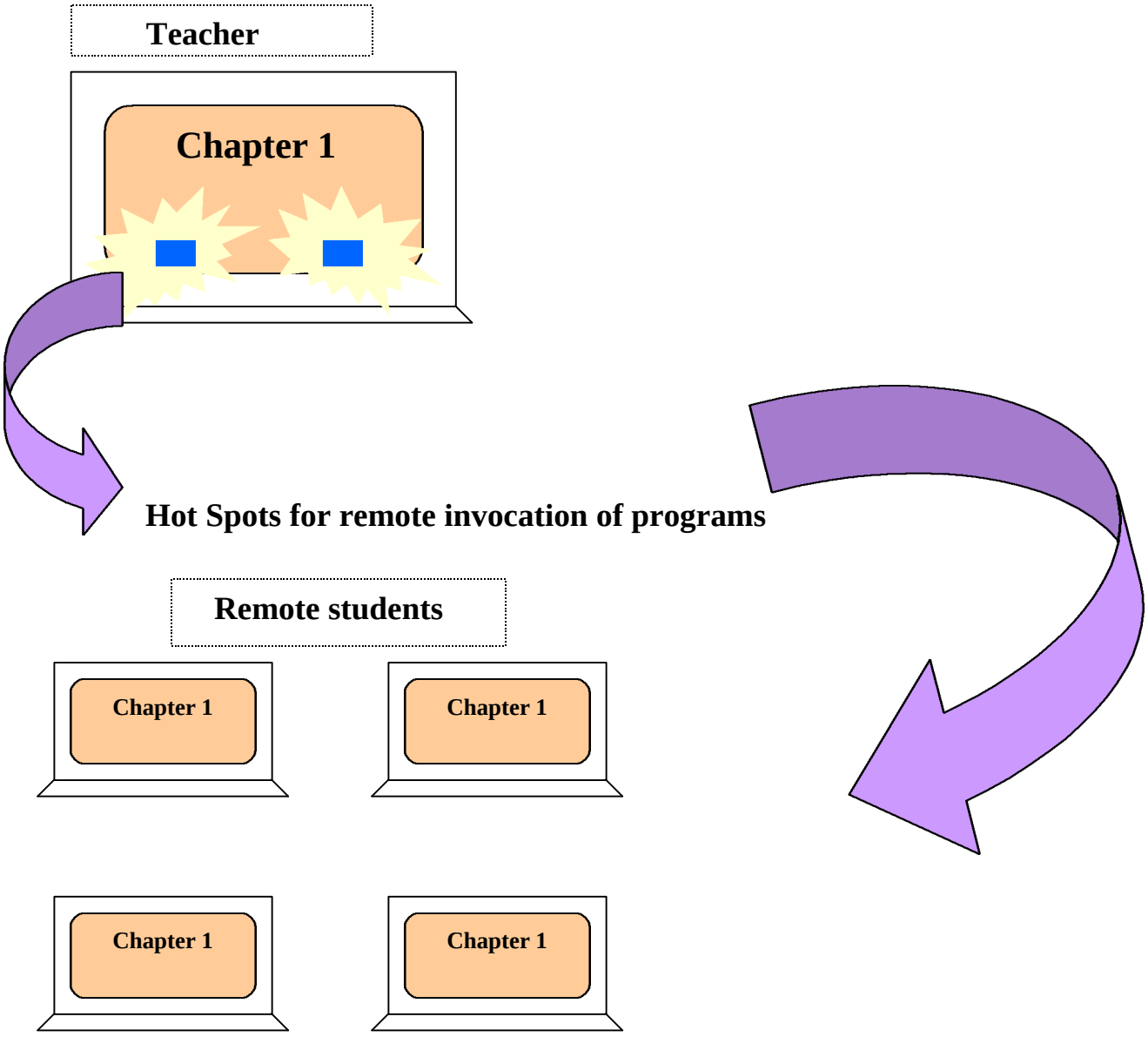


Figure 3.3

In the above figure, teacher runs the multimedia application from a CD or the local hard drive. The same application also runs on the student's machines. Hotspots (for example, buttons) in the presentation window of the application provide the navigation facility to

the teacher. When the teacher decides to go from a page to another page, he/she clicks on a hotspot. As soon as the mouse click event occurs on teacher's machine, present display instantaneously steers to a particular scene ('chapter 1' as shown in the above figure) on all the machines in the session. It is emphasized here that the near-real-time communication is possible in our case since the multimedia application is

- **not** transmitted over the network,
- **saved locally** at both ends, and
- only **remotely activated** on student's machines.

The above implementation will consist of two different versions of the course material, teacher's version or the 'Instructor CD' and the student's version or the 'Student CD'.

3.3.a Typical course operation

Registration Process :

- Students sign up for the course.
- Student provide his/her email address to the university.
- Student pays the tuition fee.
- Student's name is added to the database. Student obtains course CD (student version).

At the beginning of a semester,

- Teacher receives the email address file.
- Teacher emails IP address of his/her machine to all the remote students already registered for the course.
- Student reads and saves the IP address provided by the teacher.

3.3.b Interaction process between Instructor CD and Student CD

The interactions between the instructor program and the student program are illustrated the following flowchart.

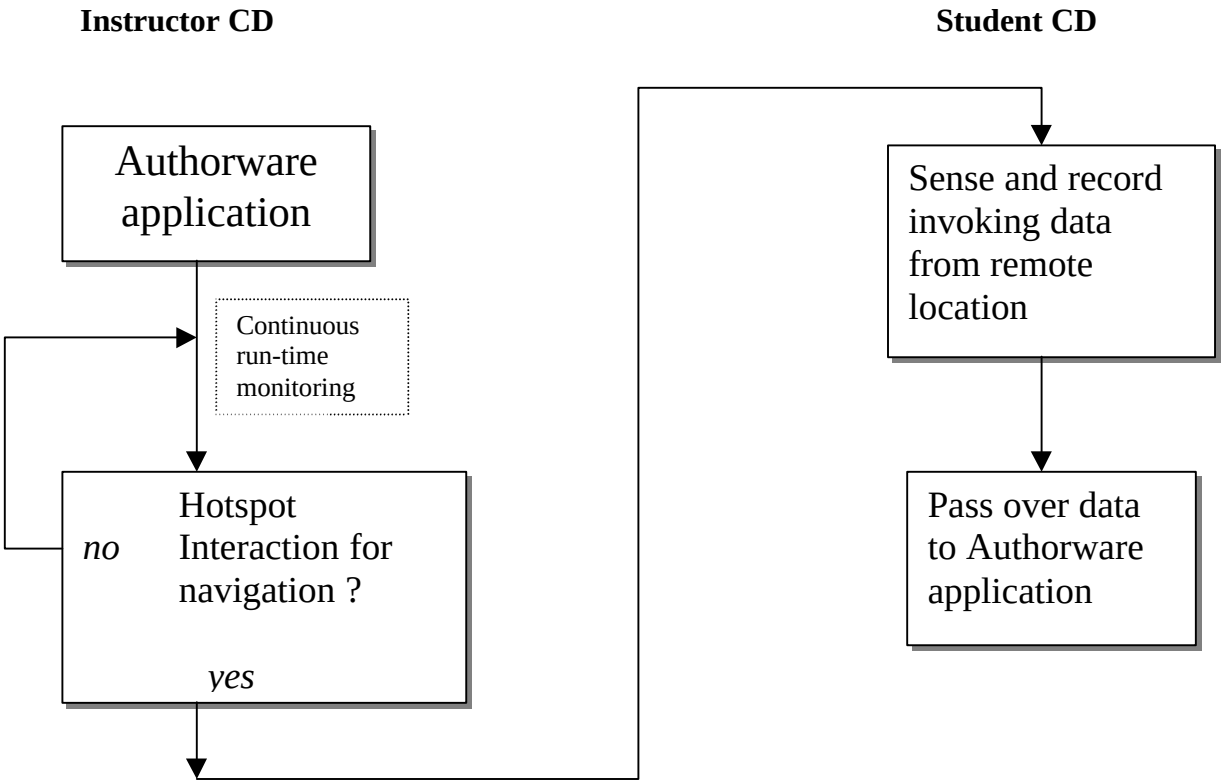


Figure 3.4

Teacher runs the Authorware application from the 'Instructor CD' to teach. At the remote locations, students insert 'Student CD' and run the program. Students then see the same material as the instructor.

3.4 System Design

System design requires developing an interface program for Authorware. This interface program allows the Authorware application to talk to the TCP/IP stack on a local machine. On the server or teacher's CD, this interface program runs the teacher's machine as a server over a TCP/IP network greeting requests from connecting student machines. This interface program is responsible for transmitting the activation data to the appropriate segments of the Authorware application running on the remote machines. On the other side, the interface program on client or student's CD is responsible for connecting to the teacher's machine, sensing and recording incoming data from server, and passing it to the local Authorware application for the desired navigation to occur.

Two implementation approaches are possible.

1. Running the interface program from within Authorware application using dynamic linked libraries (DLLs) for windows platform or XCMDs (X-commands) for Macintosh platform.
2. Running the interface program and Authorware application as stand-alone entities on a single machine.

Both approaches were pursued, and the second approach involving stand-alone programs was finally chosen, as explained in section 3.4.b.

3.4.a Introduction to DLLs and their need in authorware programs [10]

The developers of Authorware had the foresight to realize that they could not provide every possible action that will be required by users through the icon interface, due to which they have included the calculation icon. The mechanism of calculation icon allows

a user to program using the functions and variables that Authorware provides to perform many actions that are beyond the range of simple icons. Examples are opening external files, reading in data to display, keeping records of student's use and abilities etc. Although Authorware is promoted as a tool for developing courseware without requiring any programming background, quite a bit of programming is needed for advanced applications. All traditional kind of programming is done within the calculation icon.

The concept of DLL or Dynamic Linked Library is an important and extremely useful feature of the Windows environment. Without being very imprecise, a function can be considered a part of a program that performs a specific task. A function cannot be run on its own but needs to be **called** from another program, e.g. Authorware. Usually, the function is part of the program when it is written or compiled. However, a function can be **dynamically linked** from a library when the program is actually running, thus giving rise to the concept of a DLL (a function in a Dynamic link Library). As a matter of fact, much of Windows itself is in the form of DLLs stored in the files USER.EXE, GDI.EXE and KRNL?86.EXE in the windows system directory.

In theory, these DLLs could be linked and run from any program, such as a word processor or a spreadsheet. However it is not usually easy to make this link. Authorware was designed with the idea of making it extensible using DLLs and has a menu item on the data menu to load a function – called custom functions by Authorware. Once a custom function or DLL is loaded it can be used by Authorware exactly as if it were a system function. In addition to the function code, Authorware has some simple documentation, which can be seen using the show function options under the data menu. If a DLL is specifically written for Authorware and this extra documentation resource is included, it is

called a User Code Document or UCD. A UCD may be considered to be simply a DLL with built in documentation for Authorware to use. It is easier to import a UCD into Authorware because of the extra detail stored with the function that Authorware can read. Creating a custom library of routines for dynamic linkage into a windows Authorware application involves a learning curve for Windows programming. Basic theory behind windows programming is described briefly in the next section.

Windows Programming : Theory and Concepts [9]

Windows has a reputation of being easy for users but difficult for programmers with a long learning curve. It is quite common for newcomers to be baffled by the architecture of Windows and the structure of the applications that run under the system. Windows architecture is event driven. Following figure 3.5 illustrates this principle.

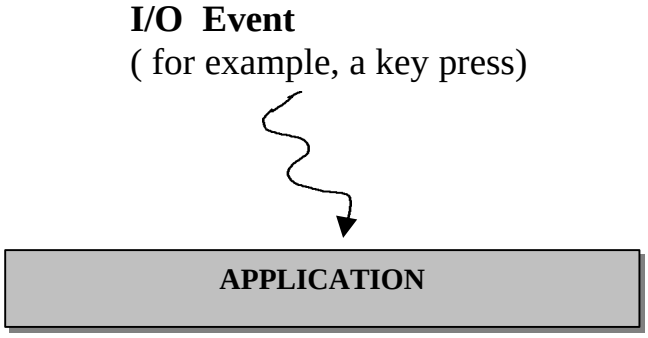


Figure 3.5

The traditional MS-DOS architecture is very well suited in a single-tasking system, where it is generally assumed that only one program at a time will be loaded into memory and running. But, in windows, there can be many different programs in memory at the same time, sharing CPU time, each one represented by a window on the screen.

As shown in the above figure, Windows is responsible for all mouse-clicks and moves, all keystrokes, all disk and printer accesses, and all I/O-related events. No single program can take direct control of an I/O device. For example, when the user presses a keyboard key, Windows figures out which window has the keyboard focus and sends the keystroke information to that window only.

Message Handling in Windows: Window Procedure

In Windows vocabulary, a message is a small unit of information. A message is sent by Windows to an application's window any time an event takes place that affects the application. Windows transmits a message to an application by calling a function in that application. This is one of the key concepts in Windows programming. A window is the key unit of program organization in Windows. So there is one message-receiving function associated with each window class. Such a message-receiving function is called a window procedure. The following figure 3.6 shows how messages are sent to window procedures.

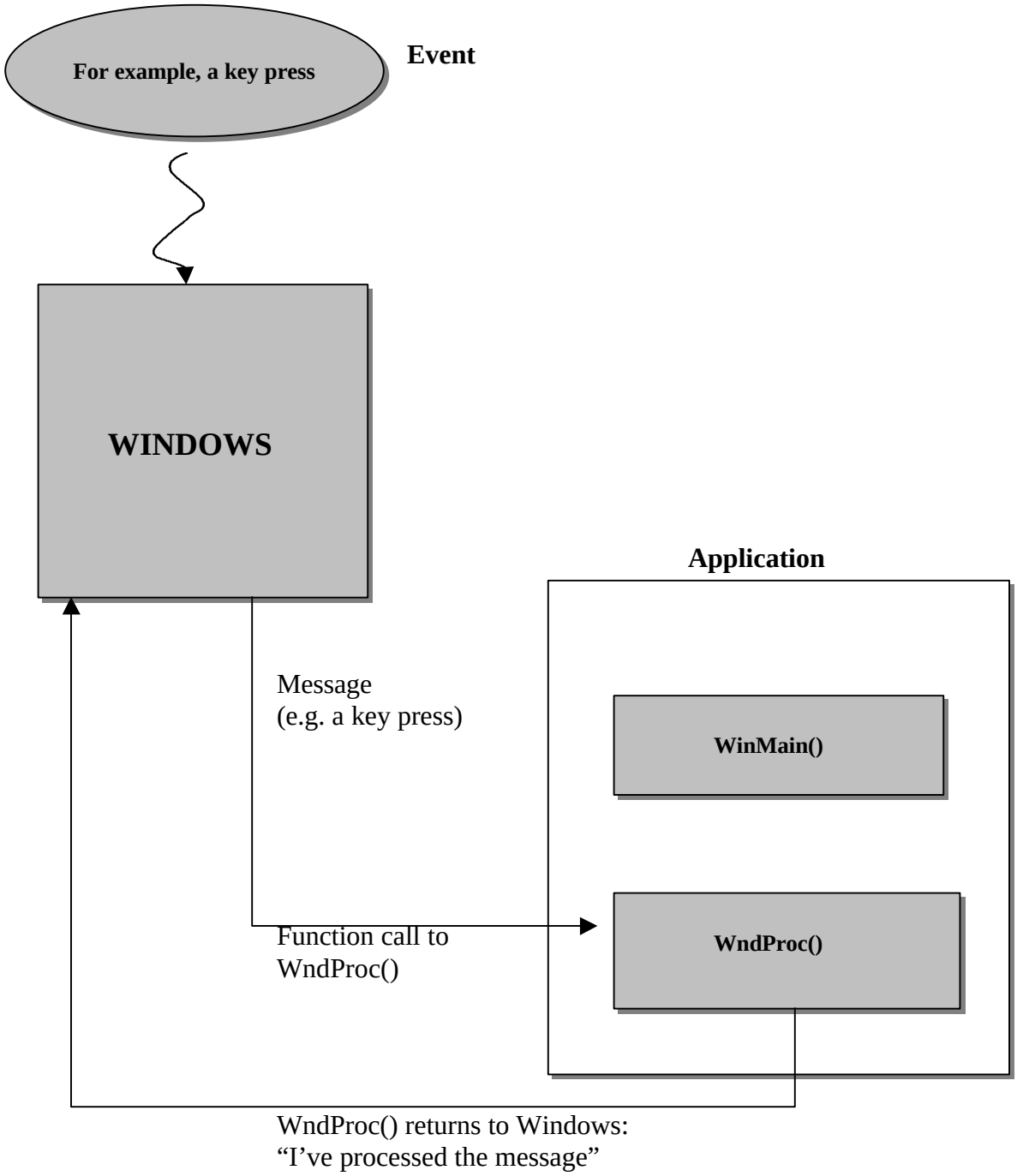

For example, a key press
Event
WINDOWS
Application
Message
(e.g. a key press)
WinMain()
WndProc()
Function call to
WndProc()
WndProc() returns to Windows:
"I've processed the message"

Figure 3.6

A window procedure can have any name as long as it doesn't conflict with another name in use. A windows program can contain more than one window procedure.

Creating custom DLLs for Authorware

Authorware has the capability of importing DLLs and thus extending its functionality. Import of a DLL is done through the calculation icon of Authorware. The following figure 3.7 shows a typical DLL life cycle and its relationship with Authorware program.

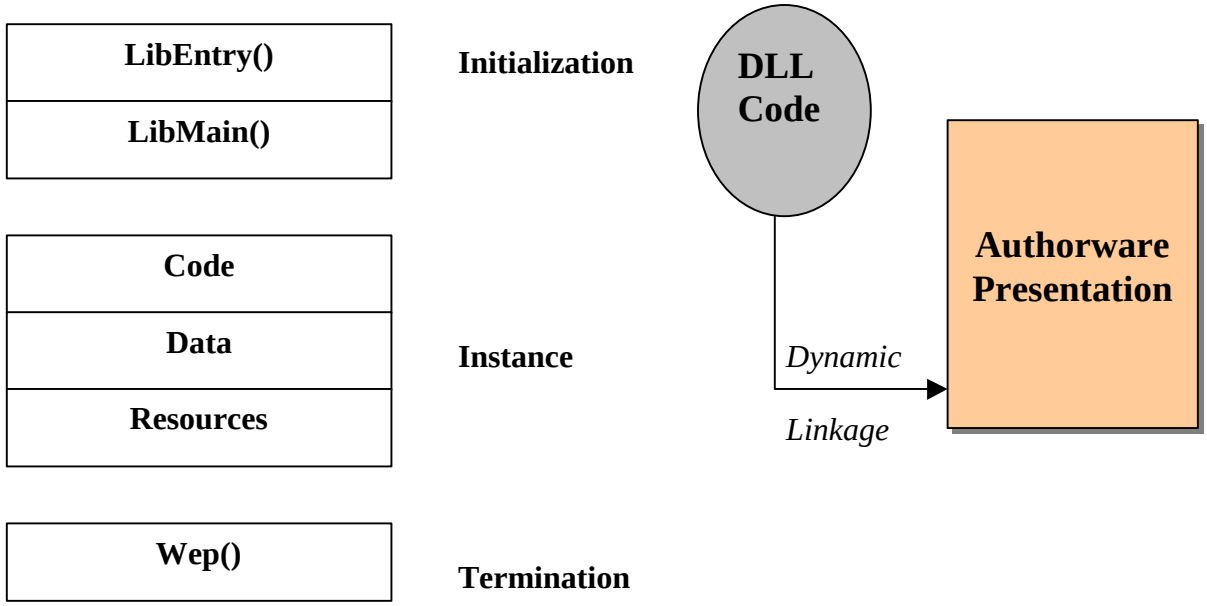


Figure 3.7

The following discussion describes how to write a DLL for the 32-bit windows platform. Three files need to be created in order to successfully import a DLL into Authorware program. They are described in details below.

1. The windows definition file which defines heap size etc. This file has the extension .DEF and a general example is shown below.

```
;------------------------------------
; MYDLL.DEF module definition file
;------------------------------------

LIBRARY         mydll
DESCRIPTION     mydll DLL
EXETYPE         WINDOWS
STUB            'WINSTUB.EXE'
CODE            PRELOAD MOVEABLE DISCARDABLE
DATA            PRELOAD MOVEABLE SINGLE
HEAPSIZE        1024

EXPORTS
        WEP             @1 RESIDENTNAME
        myFunction      @2
```

This compiles to a DLL library hence the name **LIBRARY**. The list of exported functions can be expanded if there are more than one function in the file. A DLL uses the calling program stack and so there is no need to specify stack size. There is a function **myFunction** in the source code that is being exported. The function WEP must be exported. The numbers are used as an index into the DLL to pick up the actual function.

2. The resource file which allows import into Authorware. This resource file has the extension .rc. The .rc file is compiled to produce a .res file before being linked with the object code into the final DLL. An example for the DLL **myFunction** is:

```
1 DLL_HEADER LOADONCALL DISCARDABLE
BEGIN
    "myFunction\0",
    "\0"
END

myFunction DLL_HEADER LOADONCALL DISCARDABLE
```

```
BEGIN
   "\0",
   "W\0",
   "W\0",
   " myFunction(int) \r\n",
   "\r\n",
   "This function takes an integer and processes on it.\0"
END
```

This file is in two sections. The first section is a list of all the functions available in this particular dynamic link library. In the example there is only one function in the library. If there were more functions they would be listed after the function **myFunction** in the first part. This part ends with the ZERO (NULL) termination.

The second part of the file describes the details of the function for documentation and to allow for easy import into a windows program.

The lines are:

```
BEGIN
   "\0",                          shows code is in this file - otherwise file name
   "W\0",                         type of value returned - see later table for
                                  possible values
   "W\0",                         type of arguments
   " myFunction (int) \r\n",      description of function
   "\r\n",                        This line puts in a newline
   "This function takes an integer and returns twice its value.\0"
                                  The final line must be zero terminated
END
```

The "\r\n" combination forces a newline in the output.

The possible return and argument types are given in chapter 3 of the Authorware manual.

The main ones are listed as below :

character	Meaning	C Windows equivalent
I	signed short integer	int, short
W	unsigned short integer	unsigned int, WORD, HWND
L	signed long integer	long, LONG

P	far pointer	far*, LPSTR, LPRECT, LPPOINT
F	floating point number	float
D	double precision floating point	double
S	handle to string (for return) or far pointer to string (for argument)	LPSTR, HANDLE
V	No arguments	void, VOID

The argument types must be in upper case.

3. The source language file which is the C source code. This has the extension .c A simple example of this is as below:

```
/*************************************************************************/
/* DLL file for testing compilation                                      */
/*************************************************************************/
#include <windows.h>
int FAR PASCAL _export WEP(int nParam);
int FAR PASCAL _export myFunction(int single);
int FAR PASCAL LibMain(HANDLE hInstance, WORD wDataSeg, WORD
                                    wHeapSize,  LPSTR lpszCmdLine)
{
        if (wHeapSize > 0)
                UnlockData(0);
        return 1;
}
int FAR PASCAL _export WEP(int nParam)
{
        return 1;
}
int FAR PASCAL _export myFunction(int single)
{
         return (2 * single);
}
```

The functions must be declared as **FAR PASCAL** and the DLL functions must also be declared as **_export**. A DLL must also contain the two functions **LibMain** and **WEP**. The user part of this source code is the function **myFunction** which takes an integer value and returns double the value as an integer. (double is a C keyword and so cannot be the

function name). The types of the parameters and the return values possible were given in the previous table.

Disadvantages of using DLLs

The system implementation approach of developing DLLs was discarded as it gave rise to many complex issues. There was a problem in passing strings between a DLL and the version of Authorware used. The problem arises partly in that C/C++, which is always used to create custom method libraries, usually addresses within a 64k segment and so only requires 16 bit addresses. Since a DLL is not in the same segment, then full addresses must be forced to allow the DLL and the extended program to share variables. The other part of the problem is that windows moves around memory blocks as it runs to make the most efficient use of memory. It is therefore impossible to simply use full memory addresses with dynamic memory allocation. Authorware API documentation does not elaborate on these issues or suggest possible workarounds. Development of DLLs for the prototype model proved less crash proof along with their complexity to implement for the pursued Internet support.

Also, in the future releases of this system, when more sophistication will be required, integration of character strings with the interface development will be necessary. Among few other issues, different interface versions for different platforms need to be created. This needs the programmer to study the details of the base operating system interface and resources availability. Furthermore, creating custom library routines for dynamic linkage involves considering cross platform issues for Authorware API-specific event handling functionality. The whole process will always involve more production time and systems that will be difficult to maintain.

More technical support from Macromedia, richer API functionality with newer releases of Authorware will be necessary before the option of DLLs for future versions of this project is again considered.

3.4.b Creating standalone programs with file I/O linkage :

Standalone programs with file I/O linkage was the other approach pursued that proved to be successful. A schematic diagram of the implementation in the server side is shown in the figure 3.8 below.

Server Side

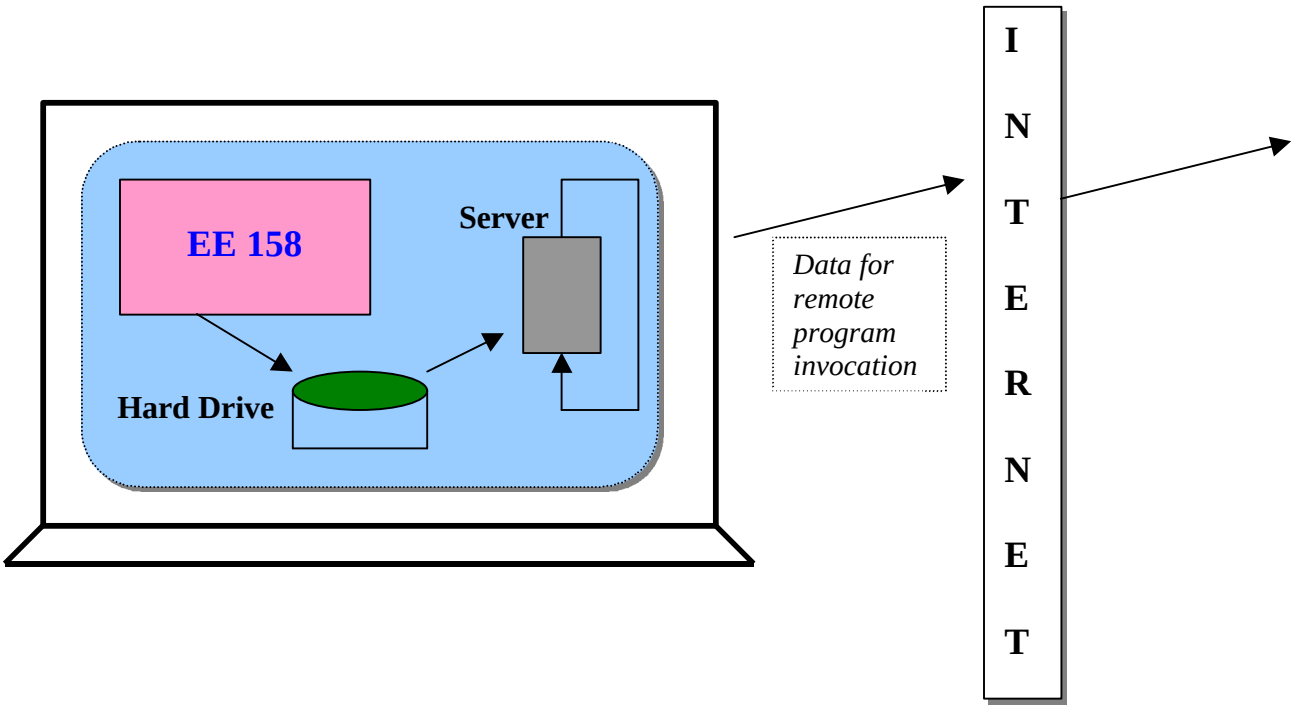


Figure 3.8

The Authorware application runs on the teacher's machine. When the teacher elects to go from a page to another page using the associated hyperlink, Authorware program flow writes the information on the local hard disk at a specified location. The server process runs as a daemon or a background process on the machine. Server reads the piece of information from the specified location on the hard drive and sends it across the TCP/IP network to the connected clients.

Client Side

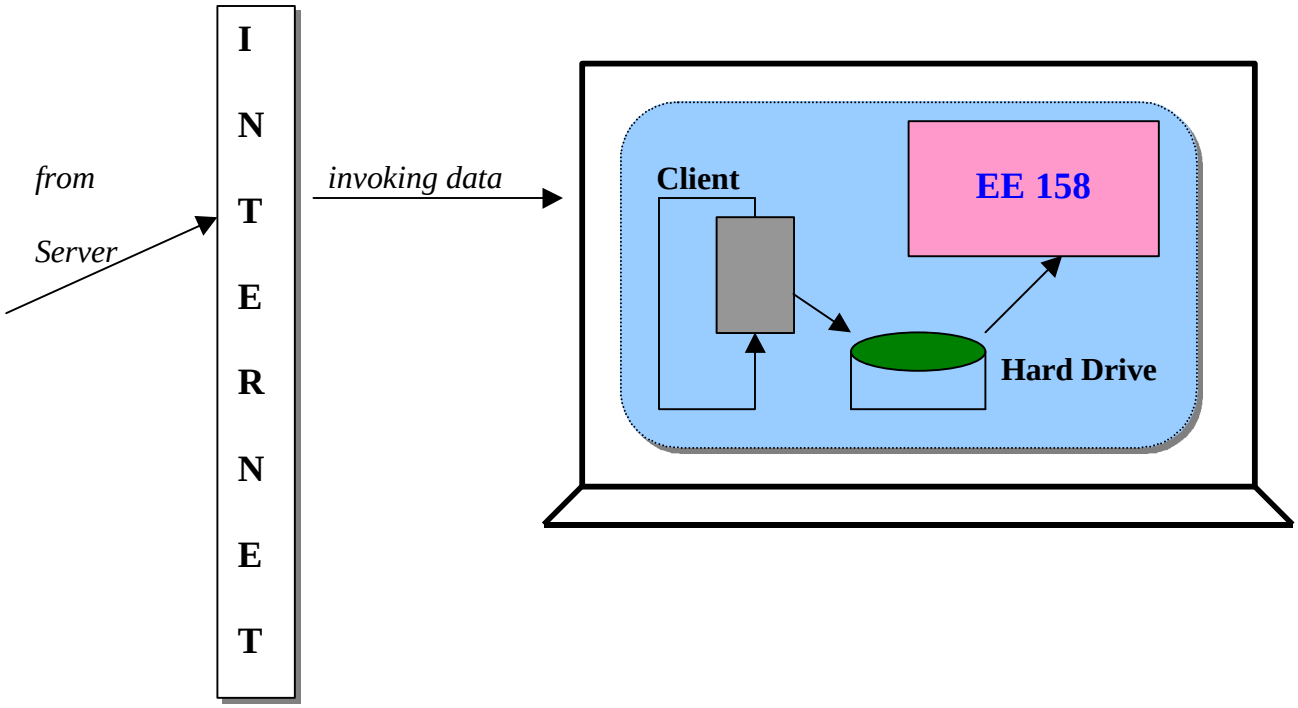


Figure 3.9

The above figure 3.9 shows the scheme implementation at the client end. The Authorware application runs on remote student's machine. The client process runs as a daemon process on the machine. The client receives the data sent by the server machine, which is

written at a specified location on the local hard drive for the Authorware program to read. For the desired navigation, the routine within the Authorware program flow is called upon. This scheme offers many advantages. Running the multimedia program separately makes it isolated from TCP/IP interface implementation and thus, easy to maintain in future versions of this system. The standalone daemon processes can be easily updated independent of the multimedia logic flow. This gets rid of the complexities offered by the implementation of dynamic linkages. System developed with stand-alone programs is observed to be more robust and crash proof during its testing phases.

Implementation details :

A. Server side

As mentioned in the previous sections, all of the student machines connect to the teacher's machine on a TCP/IP network such as the Internet. Thus, the server process on teacher's machine must be run as a background process (or daemon process). It must also allow any number of clients to connect to the machine it is running on. This specifies that the implementation of the server in software program be concurrent, multithreaded and reliable. The server is programmed to be a concurrent, connection oriented (see Appendix B for details) server. Multithreaded nature of the server process is shown in the following figure 3.10 [11].

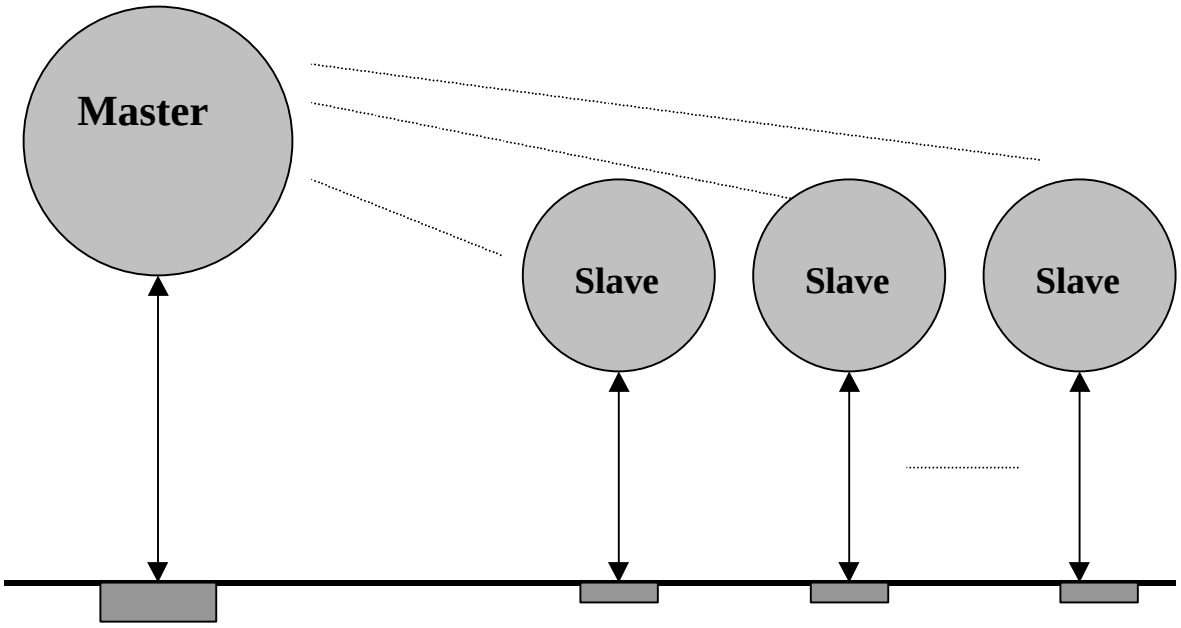


Figure 3.10

As the figure 3.10 shows, the master server thread does not communicate with clients directly. Instead, it waits at the well-known port for the next connection request. Once a request arrives, the system returns the socket descriptor (see Appendix C for socket descriptor details) of the new socket to use for that connection. The master server thread creates a slave thread to handle the connection, and allows the slave to operate concurrently. At any time, the server consists of one master thread and zero or more slave threads. To avoid using CPU resources while it waits for connections, the master thread uses a blocking call of accept (see the server code listing next) to obtain the next connection from the well-known port. When a connection request arrives, the call to accept returns, thus allowing the master thread to execute. The master creates a slave

thread to handle the request, and reissues the call to accept. The call blocks the master thread again until another connection request arrives.

Server is implemented using C-based winsock version 2.0 calls. The following flow diagram 3.11 shows the steps in the code execution.

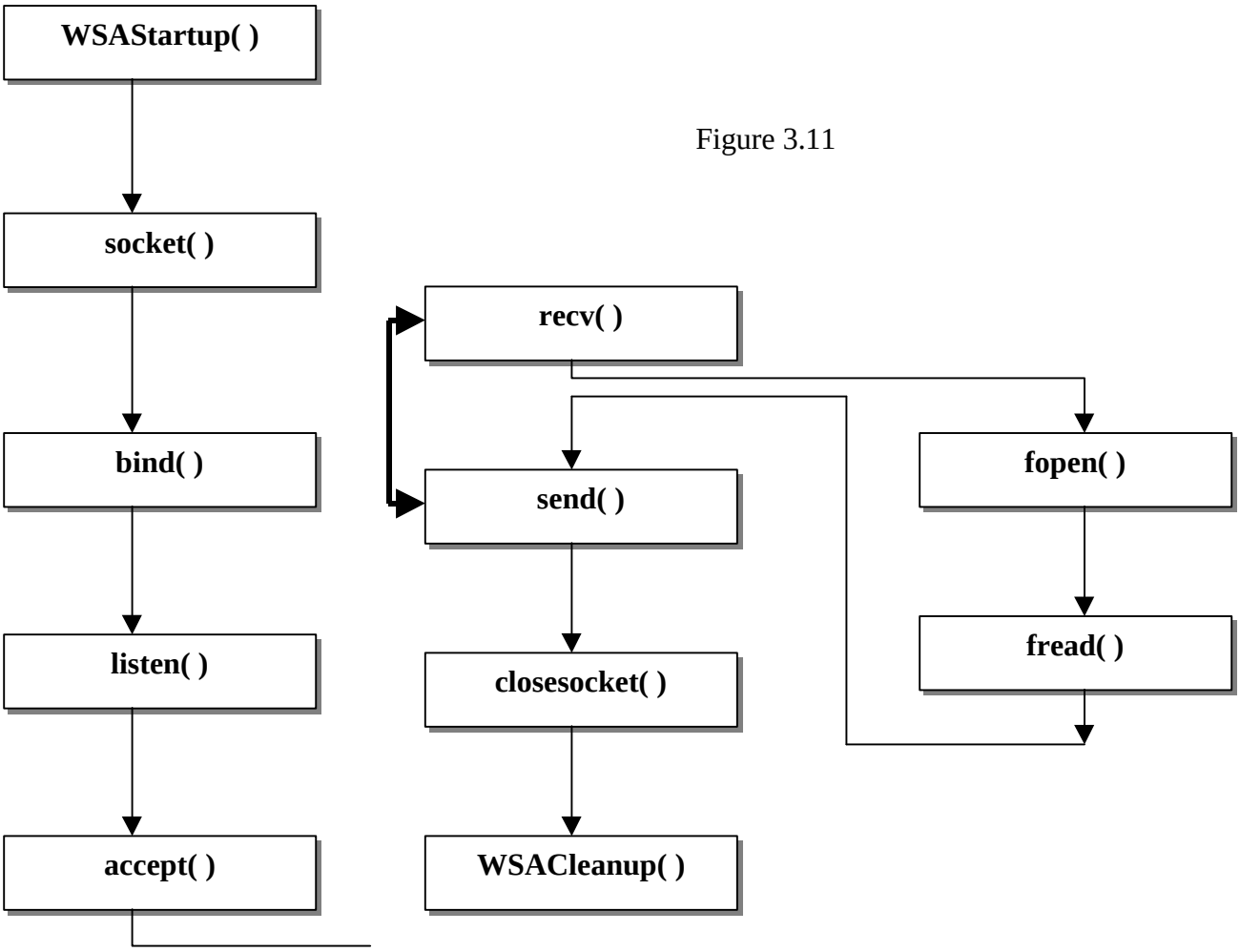


Figure 3.11

Software implemented using windows sockets must call WSAStartup before using sockets. The call requires two arguments. The program uses the first to specify the version of Windows Sockets that is requested; the operating system uses the second to return the information about the version of Windows Sockets actually used. The first argument is an

integer that gives the version number in hexadecimal. The second argument points to a WSADATA structure into which the operating system writes version number. WSAStartup is needed with the Windows operating system because the system uses dynamic linked libraries. Thus, instead of hard wiring code into the operating system, such systems bind to a version of the code at run-time. Socket call along with necessary arguments creates a new socket for network communication. The call returns a descriptor for the newly created socket. Call to 'bind' specifies the local end point address for a socket. The bind call takes arguments that specify a socket descriptor and an end point address. Connection-oriented servers then call 'listen' to place a socket in passive mode and make it ready to accept incoming requests. For TCP sockets, the server calls 'accept' to extract the next incoming connection request. An argument to accept specifies the socket from which a connection should be accepted. Servers use 'recv' to receive data from a TCP connection. Usually, after a connection has been established, the server uses recv to receive a request that client sends by calling 'send'. Servers then use 'send' to transmit replies. 'send' copies outgoing data into buffers in the operating system kernel, and allows the application to continue execution while it transmits the data across the network. Once server finishes using a socket, it calls 'closesocket' to deallocate it. If several processes share a socket, closesocket decrements a reference count and deallocates the socket when the reference count reaches zero. WSACleanup is called to deallocate all data structures and socket bindings.

'fopen' and 'fread' calls accomplish the desired file I/O. A specified file on the local hard drive is opened and read into a buffer initialized to null. Null buffer initialization is essential to make sure that no invalid data is sent to clients. Updated data is sent to all the

connected clients. ‘send’ and ‘recv’ calls execute in a continuous loop until the application is gracefully closed.

It is important to note that before an application program written in C can use the predefined structures and symbolic constants associated with sockets, it must include a file that defines them. In this system, client and server programs use C-based winsock2 API calls and hence include <winsock2.h>.

Algorithms and Issues in Server Software Design [11]

Conceptually, a server consists of a simple algorithm that iterates forever, waiting for the next request from a client, handling the request, and sending a reply. In practice, however, servers use a variety of implementations to achieve reliability, flexibility, and efficiency.

Server Types

a. Iterative, Connectionless Server

The most common form of connectionless server, used especially for services that require a trivial amount of processing for each request. Iterative servers are often less susceptible to failures.

b. Iterative, Connection-Oriented Server

A less common server type used for services that require a trivial amount of processing for each request, but for which reliable transport is necessary. Because the overhead associated with establishing and terminating connections can be high, the average response time can be non-trivial.

c. Concurrent, Connectionless Server

An uncommon type in which the server creates a new thread to handle each request. On many systems, the added cost of thread creation dominates the added efficiency gained from concurrency. To justify concurrency, either the time required to create a new thread must be significantly less than the time required to compute a response or concurrent requests must be able to use many I/O devices simultaneously.

d. Concurrent, Connection-Oriented Server

The most general type of server because it offers reliable transport (i.e., it can be used across a wide area internet) as well as the ability to handle multiple requests concurrently. Two basic implementations exist: the most common implementation uses concurrent threads or processes to handle connections; a far less common implementation relies on a single thread and asynchronous I/O to handle multiple connections.

System developed in this thesis project uses concurrent threads to handle connections.

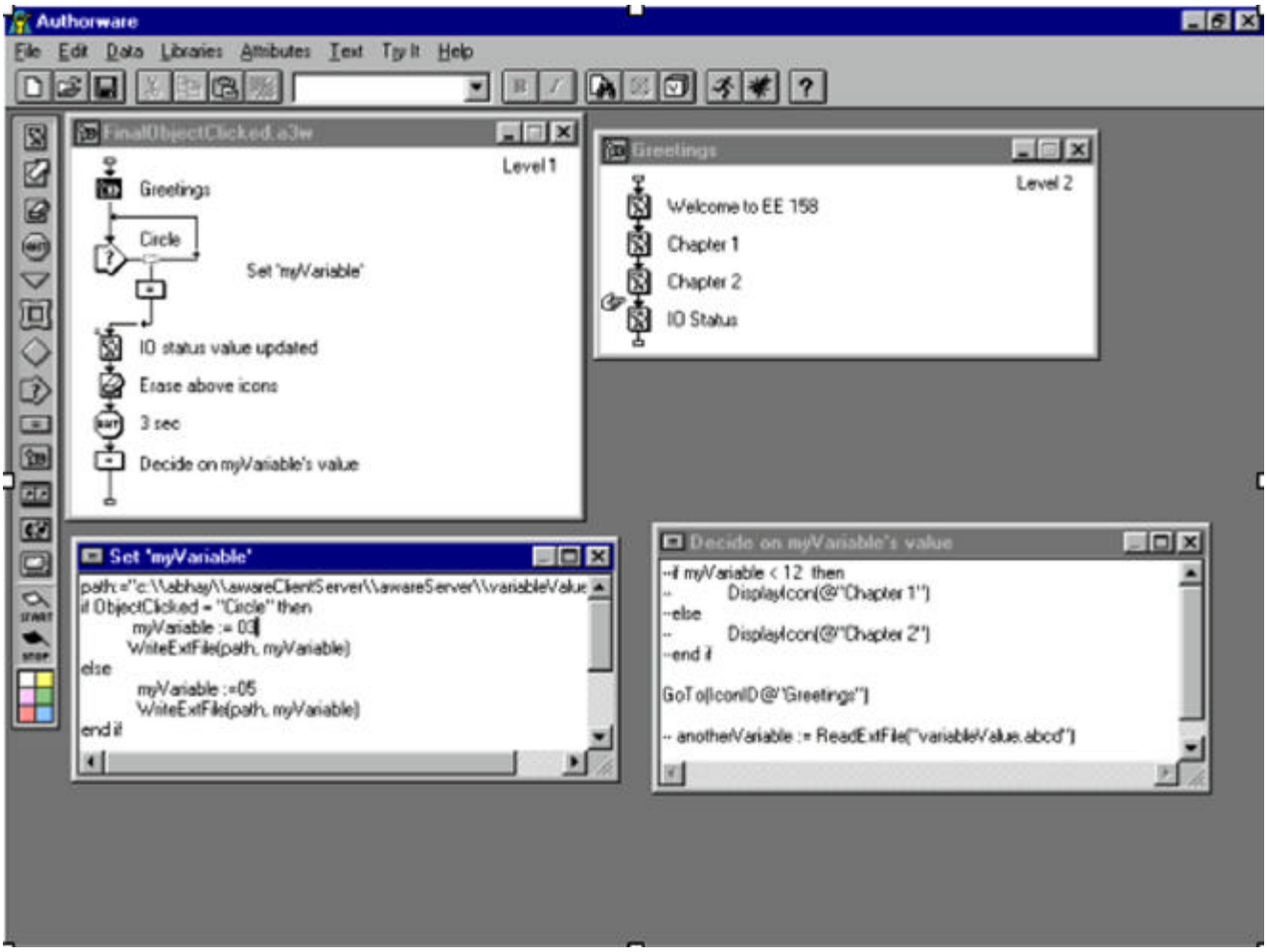
In a concurrent process implementation, the master server process creates a slave process to handle each connection. Each process has its own address space and cannot share data among slave processes.

In a concurrent thread implementation, the master thread creates slave threads within the same process to handle each connection. All of the threads share the same process to handle each connection. All of the threads share the same global address space and can share data.

Iterative implementations work well for services that require little computation. When using a connection-oriented transport, an iterative server handles one connection at a time; for a connectionless transport, an iterative server handles one request at a time.

To achieve efficiency, servers often provide concurrent service by handling multiple requests at the same time. A connection-oriented server provides for concurrency among connections by creating a thread or process to handle each new request.

When the Authorware application is running, the position of the cursor is continuously monitored by the Authorware run-time. When a hyperlink associated with an icon is encountered within the boundaries of the presentation window, and is selected by the user (Teacher, in this case), the corresponding file I/O is done and the information in the form of a text file is recorded and made available for the server process. The following snap shot shows the logic flow in Authorware code lay out.



{ If the above image is not clear in the electronic format, reader is kindly advised to refer to the hard copy saved with Dept. of Computer Science and Electrical Engineering, West Virginia University }

B. Client side

Client is implemented using C-based winsock version 2.0 calls. The following flow diagram 3.12 shows the steps in the code execution.

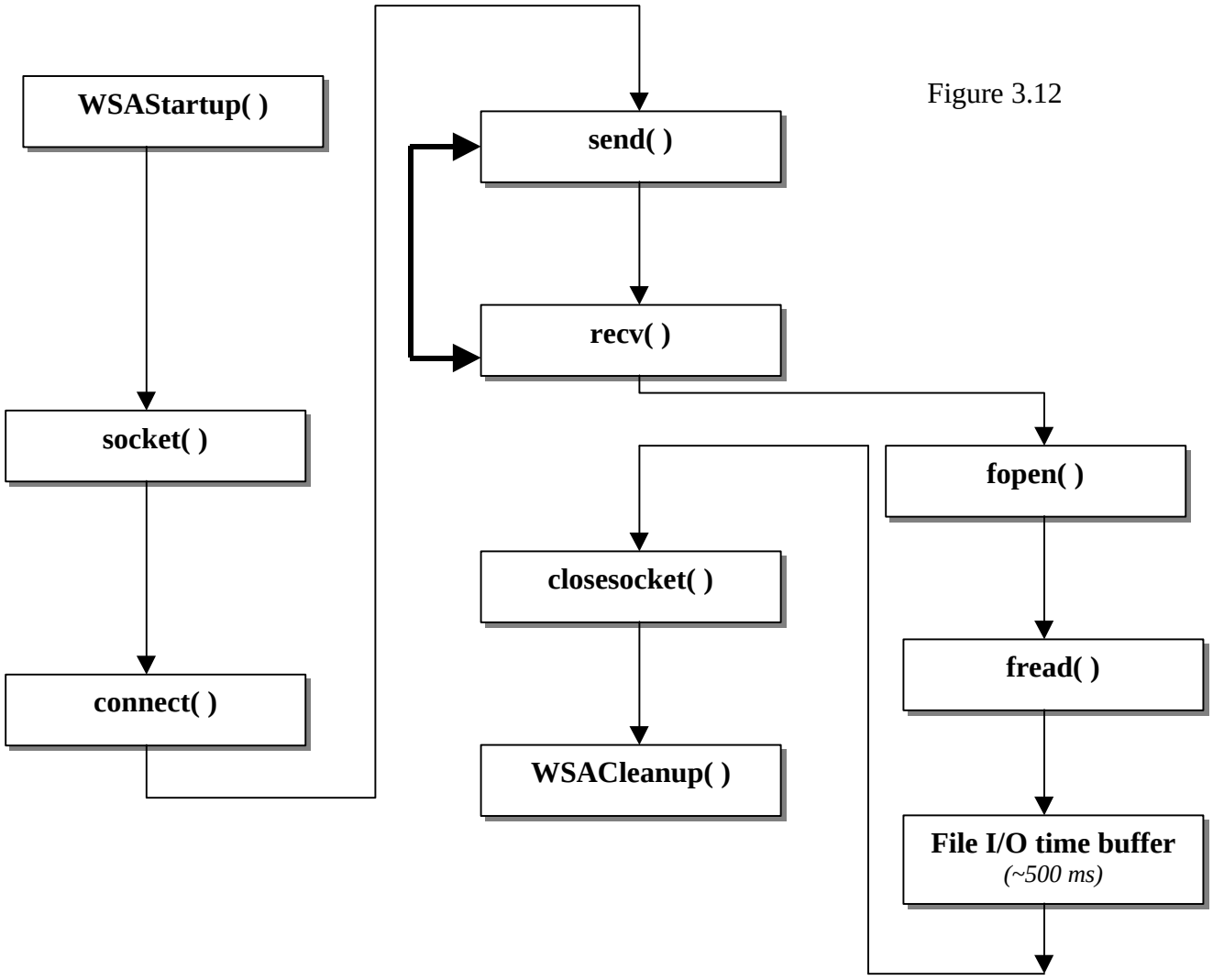


Figure 3.12

After creating a socket, a client calls connect to establish an active connection to a remote server. An argument to connect allows the client to specify the remote endpoint, which includes the remote machine's IP address and protocol port number. Client also uses

'recv' to receive a data from a TCP connection. Client daemon process receives the updated data over network and saves it on the local hard drive at a specified location. The desired file I/O is done using 'fopen' and 'fread' calls as shown in the above flow diagram.

Importance of introducing file I/O time buffer :

System, when tested without any time buffer, was observed to crash frequently. With every function call and the overall logic working correctly, the primary reason behind these crashes was found to be in the race conditions occurring on the client machines. During test cases, client machine platform and configuration varied. The local machine CPU speed and the bus architecture for file I/O on every machine can differ. As a workaround, a general time buffer of 500 *ms* is introduced after the new data is saved on the hard drive before the multimedia application reads it. This solution gave encouraging results with less number of failures.

The introduction of time buffer does not offer optimum and perfect solution. In future, the system developer is advised to give more serious consideration to this issue in his/her attempts to make the system more robust and performance tuned.

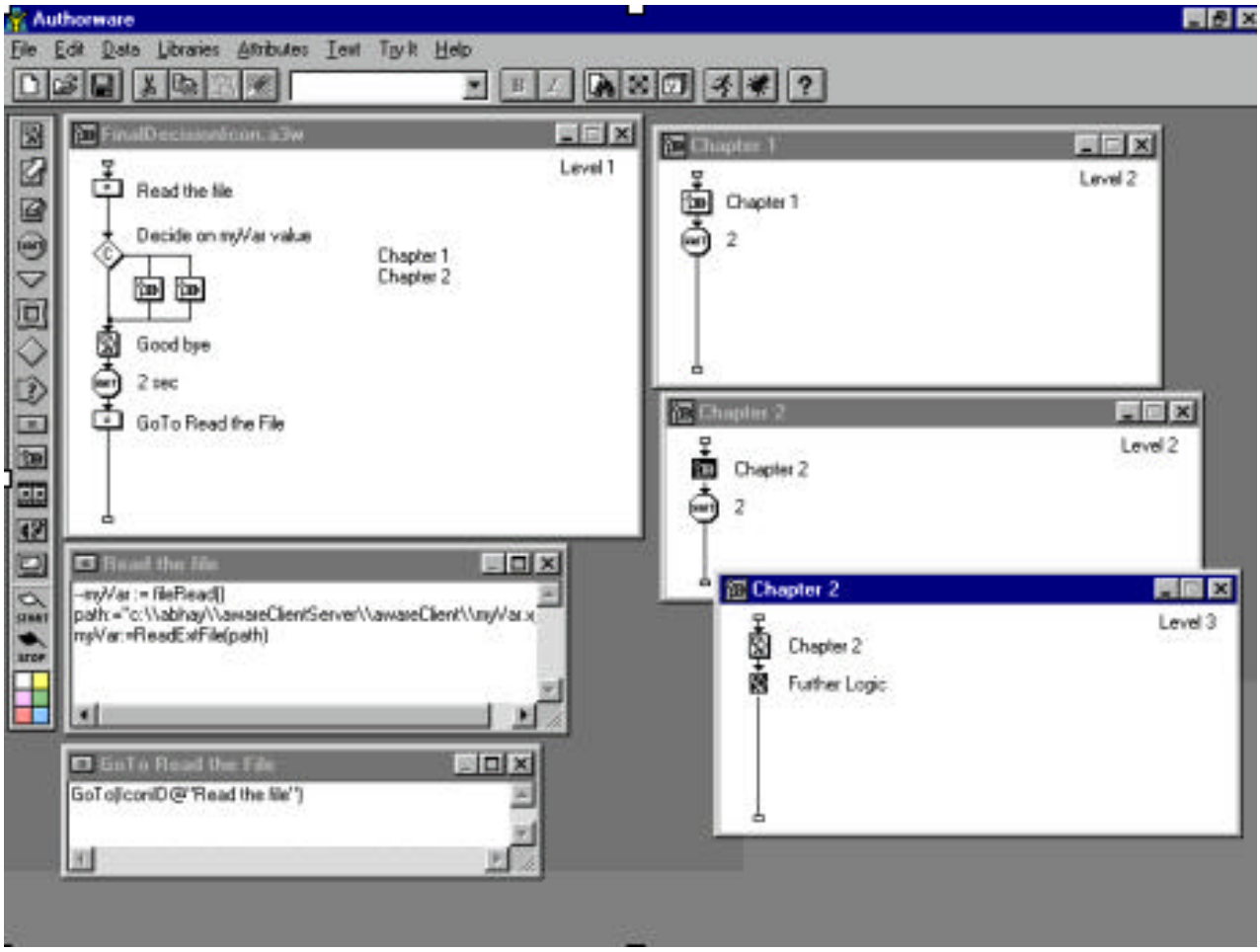
Algorithms and Issues in Client Software Design [11]

The client software program must obtain the server's IP address and protocol port number before it can communicate; to increase flexibility, client programs often require the user to identify the server when invoking the client. The client then converts the server's address

from dotted decimal notation into binary, uses the domain name system to convert from a textual machine name into an IP address.

Although a client must explicitly specify the endpoint address of the server with which it wishes to communicate, it can allow TCP/IP software to choose an unused protocol port number and to fill in the correct local IP address. Doing so avoids the problem that can arise on a gateway (router) or multi-homed host when a client inadvertently chooses an IP address that differs from the IP address of the interface over which IP routes the traffic.

Thus, the client process is now connected to the teacher's server machine. After it saves the data on the local hard drive, it is made accessible for the multimedia application. Depending on the value read from the hard disk, navigation on the client machine is decided. This is shown in the following snapshot of Authorware code layout.



{ If the above image is not clear in the electronic format, reader is kindly advised to refer to the hard copy saved with Dept. of Computer Science and Electrical Engineering, West Virginia University }

3.5 Test Cases

The system was tested on 32-bit windows machines with winsock version 2.0 run-time installed. Tests were conducted at three different times on weekdays. Results are tabulated as below and are described in details in the next chapter.

Case I

Server and Client Location	:	Engr. Sciences Bldg. Lab 813
Test Conducted on	:	Tue July 14, 98 9:30 p.m.
Operating Platform	:	Windows NT 4.0
Server machine	:	157.182.81.96
Successful Client machines	:	157.182.81.94 157.182.81.95 157.182.81.96 157.182.81.97 157.182.81.98 and 3 dynamic IP machines
Failed cases (after 3-4 minutes)	:	157.182.81.232 157.182.81.234
Time to invoke	:	very small (near real-time)

Case II

Server Location	:	Engr. Sciences Bldg. Lab 813
Client Location	:	Engr. Sciences Bldg. Lab 231/249
Test Conducted on	:	Wed July 15, 98 2:45 p.m.
Operating Platform	:	Windows NT 4.0 / Windows 95
Server machine	:	157.182.81.96 (Win NT)

Successful Clients	:	157.182.81.53 (Win 95)
		157.182.81.56 (Win 95)
		157.182.81.96 (Win NT)
Time to invoke	:	very small (near real-time)

Case III

Server Location	:	Engr. Sciences Bldg. Lab 813
Client Location	:	Engr. Sciences Bldg. Lab 813 +
		Engr. Research Bldg. 209
Test Conducted on	:	Wed July 15, 98 9:10 p.m.
Operating Platform	:	Windows NT 4.0 / Windows 95
Server machine	:	157.182.81.96 (Win NT)
Successful Clients	:	157.182.196.19 (Win 95)
		157.182.81.23 (Win 95)
		157.182.81.96 (Win NT)
Time to invoke	:	very small (near real-time)

Chapter 4 Conclusions and Future Work

4.1 Discussion

The results of the preliminary tests carried out are listed in chapter 3. Tests were carried out at three different times on weekdays, however the server and client machines all belonged to the same network. Time for the invoking data to reach participating client/student machines was very small in all the three cases. Data arrived at client machines was not corrupted (since the queuing buffers used to temporarily save the data are null-initialized), thus ensuring accurate and synchronous navigation on student machines. The results of the preliminary tests confirm the feasibility of the proposed approach and the methodology described in this thesis. The system developed can be successfully used for distance education applications.

The two failed instances listed in Case I were both Pentium 90 MHz machines as against the successful clients which were Pentium 200 MHz machines. The anticipated reason behind these failures is the varying file I/O time period on the local hard drive, giving rise to a race condition as described in detail in chapter 3. Increased file I/O time buffer may help reduce the number of failures but a more sophisticated and invariable solution may be expected from developers of future versions of the current system.

Pre-installation of winsock 2.0 (or later) run-time is expected on all the machines before the system is run. This run-time software is free to download from Microsoft's official site or may also be provided to students on their CDs.

The system developed is also very general and hence, compatible with any multimedia authoring tool (like Authorware) available in market today. This authoring software package is expected to have built-in support for the required file I/O.

4.2 Future Work

The current system operates on PC 32-bit windows platform. (Further tests on windows 98 operating system platform are not expected to give rise to any surprising hurdles.)

With appropriate translation of existing winsock API calls, this system should be easily portable to operate in UNIX or Macintosh environments. The authoring tool's availability on either platform is expected.

The daemon processes, in the present implementation, are run as windows console applications (and hence, they are in a way DOS based). DOS does not feature multitasking. In future versions, the processes may be upgraded to full-blown windows programs to take advantage of multitasking characteristic of windows operating system. With this facility, Instructor and student CDs can be made entirely auto-enabled. Thus, on the Instructor's machine, after insertion of the CD, both the server process and the multimedia program can be started simultaneously. On the student's machine, a dialog box interaction to enter server's IP address can be streamed with succeeding calls to client daemon process and multimedia program.

The current system is developed using Macromedia Authorware version 3.5. More refinement may be possible with the use of latest version 4.03, which has increased in-built network functionality, and easy embodiment of Java applets. This may also ensure an on-line chat and a live audio link among all active machines during the session.

Chapter 5 References

[1] “Distance Education at a Glance”, Engineering Outreach Guides, University of Idaho, Moscow, ID; ed.: Tania H. Gottschalk

[2] Moore, M. G., Thmopson, M.M., Quigley, A.B., Clark, G.C., and Goff, G.G.; “The effects of distance learning: A summary of the literature”. Research Monograph No. 2. University Park, PA: The Pennsylvania State University, American Center for the study of Distance Education. (ED 330 321), 1990.

[3] Verduin, J.R. & Clark, T.A. (1991). “Distance education: The foundations of effective practice”. Jossey-Bass Publishers, San Franscisco, CA, 1991.

[4] A. Bakshi, Dr. B.Das, “Multimedia Based Distance Education through the Internet”, Conference Paper: ‘Frontiers in Education Conference’, Pittsburgh, PA, 1997.

[5] Authorware and Director are trademark software of Macromedia Inc. *www.macromedia.com*

[6] Ginsburg M., December J., “HTML and CGI Unleashed”, first edition, pg. 5-7 1995.

[7] David Miller, “Web Multimedia Development”, pg. 13-22 1996.

[8] Macromedia's official site: 'www.macormedia.com'

[9] Charles Petzold, "Programming Windows95", Chapter 2 pg. 13-48 1996.

[10] B. Wood, Cripps Computing Center, University of Nottingham, England, 'Extending Authorware for Windows using DLLs' draft version 2.2 Oct 1994

[11] Comer D.E., Stevens D.L., "Internetworking with TCP/IP" vol III windows sockets version pg. 137 1997.

Appendix A

Server Code Listing

```
/*************************************************************************
 *  Copyright (C)  1998              :      Department of Computer Science and
 *  All Rights Reserved              :      Electrical Engineering
 *                                   :      West Virginia University, Morgantown
 *                                   :      WV 26505
 *
 *  This file can not be reproduced or copied without the prior
 *  written permission from the Department of CSEE, WVU.
 *
 * This file is a part of Master's Thesis Report submitted by
 * Abhay Bakshi under the guidance of Dr.B.Das, Assistant Professor,
 * Department of CSEE, WVU.
 *
 *  File Name                        : awareServer.cpp
 *  Linked Libraries                 :  kernel32.lib user32.lib gdi32.lib
 *                                     winspool.lib comdlg32.lib advapi32.lib
 *                                     shell32.lib ole32.lib oleaut32.lib
 *                                     uuid.lib odbc32.lib odbccp32.lib libcmt.lib
 *                                     ws2_32.lib msvcrt.lib
 *  File Description                 : This file  implements windows sockets 2.0
 *                                     API to run a multithreaded server on a
 *                                    windows machine. This is a concurrent
 *                                    connectionless server which waits for
 *                                    client requests, and supports overall
 *                                    communication protocol interface defined
 *                                     between an Authorware server machine and
 *                                    connecting client machines.
 *  Platform                        : Windows 32-bit console application
 *  Interface with                  : Macromedia Authorware :   ver 3.5
 *                                    Educational Edition
 *  Author                          : Abhay Bakshi
 *************************************************************************/


/*************************************************************************
 Header files
 *************************************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <winsock2.h>
#include <process.h>
```

```
/************************************************************************
 Constants to be used later in the code
 ************************************************************************/
#ifndef INADDR_NONE
#define INADDR_NONE     0xffffffff
#endif

#define WSVERS          MAKEWORD(2, 0)

#defineAWARE_SERV_TCP_PORT          7250    /* comm. port for awareClient */

struct message
{
        int xPos;
        int yPos;
};

#define QLEN                    4               /* Max queue len to listen for */
#define STKSIZE       16536

typedefstruct sockaddr_in SocketAddress;

/************************************************************************
 External variables
 ************************************************************************/
extern int errno;                       /* auto set in case of error */



/************************************************************************
 Function prototypes
 ************************************************************************/
int errexit(const char *format, ...);
int awareWorker(SOCKET slaveSocket);


/************************************************************************
 errexit()      :       Print error message and exit
 ************************************************************************/
int errexit(const char *format, ...)
{
        va_list args;

        va_start(args, format);
        vfprintf(stderr, format, args);
        va_end(args);
```

```
        exit(1);
        return 0;
}


/*************************************************************************
 main()
 *************************************************************************/
int main (void)
{
        /* tcp/ip related stuff */
        SOCKET      masterSocket=0;
        SOCKET      slaveSocket=0;

        SocketAddressawareServer, awareClient;

        int      clientAddressLength=0;
        int type=0;

   WSADATA wsadata;

        /*
         * check for the version of the winsock and exit if not the
         * the current/required version.
         */

      if (WSAStartup(WSVERS, &wsadata) != 0)
           errexit("WSAStartup failed\n");


        /*
         * Allocate a TCP socket for client connection
         */
        masterSocket = socket(PF_INET, SOCK_STREAM, 0);
        if ( masterSocket == INVALID_SOCKET)
                errexit("awareServer: can't create socket: %s",WSAGetLastError());


   /*
    * Bind our local address so that the client can send to us.
    */

        memset(&awareServer,0, sizeof(awareServer));
        awareServer.sin_family      = AF_INET;
        awareServer.sin_addr.s_addr = htonl(INADDR_ANY);
        awareServer.sin_port        = htons(AWARE_SERV_TCP_PORT);
```

```
        if (bind(masterSocket, (struct sockaddr *) &awareServer, sizeof(awareServer)) ==
SOCKET_ERROR)
                errexit("awareServer: can't bind to TCP port: %s",WSAGetLastError());

                type = SOCK_STREAM;

  /*
   * Listen on our local socket for the client
   */

        if (type==SOCK_STREAM && listen(masterSocket, QLEN) ==
SOCKET_ERROR)
                errexit("awareServer Can't listen on TCP port:%s\n",WSAGetLastError());


        while(1)
        {
        clientAddressLength = sizeof(awareClient);

                /* accept a connection when ready */

                slaveSocket = accept(masterSocket, (struct sockaddr *)&awareClient,
&clientAddressLength);
                if (slaveSocket == INVALID_SOCKET)
                errexit("accept: %s\n", WSAGetLastError());

                if(_beginthread((void (*)(void *)) awareWorker,
STKSIZE,(void*)slaveSocket)<0)
                errexit("_beginthread: %s\n",strerror(errno));

        }
        return 0;
}


/************************************************************************
 awareWorker()
 ************************************************************************/
int x=0,y=0;
FILE *stream;

int awareWorker(SOCKET slaveSocket)
{
        int cc;
        static  char list[2];           // initialize it to 0
```

```
        struct message messageBuf,tempBuf;

        /* for synchronization get signal first to send */
        /* receive the message */
        cc = recv (slaveSocket, (char *)&tempBuf, sizeof tempBuf, 0);

        stream = fopen("variableValue.abcd", "r");    /* */
        fread(list, sizeof (char), 2, stream);        /* */
        fclose(stream);

        while(cc != SOCKET_ERROR && cc > 0)
        {

                messageBuf.xPos = ntohs(tempBuf.xPos);
                messageBuf.yPos = ntohs(tempBuf.yPos);

                fprintf(stdout, "received stuff from Client\n");

                /* initialize messagebuf and do htons */
                tempBuf.xPos=htons(x++);
                tempBuf.yPos=htons(y++);

                /* send ack/echo back */
                if(send(slaveSocket, list, strlen(list), 0) == SOCKET_ERROR)
                {

                        fprintf(stderr, "echo send error: %d\n",WSAGetLastError());
                        break;

                }

                /*
                        block on next receive  and send updated file contents for every
                        client request
                */
                cc = recv(slaveSocket, (char *)&tempBuf, sizeof tempBuf, 0);
                stream = fopen("variableValue.abcd", "r");    /* */
                fread(list, sizeof (char), 2, stream);        /* */
                fclose(stream);                               /* */


        }
        return 0;
}
```

Appendix B

Client Code Listing

```
/*************************************************************************
 *  Copyright (C)  1998              :      Department of Computer Science and
 * All Rights Reserved               :      Electrical Engineering
 *                                   :      West Virginia University, Morgantown
 *                                   :      WV 26505
 *
 *  This file can not be reproduced or copied without the prior
 *  written permission from the Department of CSEE, WVU.
 *
 * This file is a part of Master's Thesis Report submitted by
 * Abhay Bakshi under the guidance of Dr.B.Das, Assistant Professor,
 * Department of CSEE, WVU.
 *
 *  File Name                        :      awareClient.cpp
 *  Linked Libraries                 :  kernel32.lib user32.lib gdi32.lib
 *                                      winspool.lib comdlg32.lib advapi32.lib
 *                                      shell32.lib ole32.lib oleaut32.lib
 *                                      uuid.lib odbc32.lib odbccp32.lib libcmt.lib
 *                                      ws2_32.lib msvcrt.lib
 *  File Description                 : This file implements windows sockets 2.0
 *                                     and reads in the file sent by the Authorware
 *                                     server machine and saves it to be read by
 *                                     Authorware client software piece.
 *  Platform                         : Windows 32-bit console application
 *  Interface with                   : Macromedia Authorware :   ver 3.5
 *                                   : Educational Edition
 *  Author                           : Abhay Bakshi
 *************************************************************************/


/*************************************************************************
 Header files
 *************************************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <winsock2.h>


/*************************************************************************
 Constants to be used later in the code
 *************************************************************************/
#ifndef INADDR_NONE
#defineINADDR_NONE      0xffffffff
#endif
```

```
#defineWSVERS                MAKEWORD(2, 0)

#define AWARE_SERV_IP_ADDR          "157.182.81.96" //      "127.0.0.1"
                                                            /* */
#define AWARE_SERV_TCP_PORT         7250

struct message
{
        int xPos;
        int yPos;
};

typedef struct sockaddr_in SocketAddress;

SOCKET        awareSocket;

struct message messageBuf,tempBuf;


/*************************************************************************
 Function prototypes
 *************************************************************************/
void    errexit(const char *, ...);
void    awareMessage(void);


/*************************************************************************
 External variables
 *************************************************************************/
extern int errno;                       /* auto set in case of error */


/*************************************************************************
 main()
 *************************************************************************/
void main(void)
{
        SocketAddressawareServer,awareClient;
        WSADATA    wsadata;

        static char buffer[20];          /* */
        FILE    *file;

        if (WSAStartup(WSVERS, &wsadata) != 0)
                errexit("WSAStartup failed\n");
```

```
        if ( (file=fopen("c:/abhay/IPaddress.txt", "r")) != NULL )
        /* */
        {
                fread(buffer, sizeof (char), 20, file);
                /* */
                fclose(file);
                                        /* */
        }
        else
                                                /* */
                printf("awareClient:main:: Error opening IPaddress.txt file\n");       /* */

        /*
         * Fill in the structure "awareServer" with the address of the
         * server that we want to send to.
         */

        memset(&awareServer, 0, sizeof(awareServer));
        awareServer.sin_family      = AF_INET;
        awareServer.sin_port = htons(AWARE_SERV_TCP_PORT);
        awareServer.sin_addr.s_addr  = inet_addr(buffer);
//      awareServer.sin_addr.s_addr  = inet_addr(AWARE_SERV_IP_ADDR);

        /*
         * Open a TCP socket
         */

        if ( (awareSocket = socket(PF_INET, SOCK_STREAM, 0)) ==
INVALID_SOCKET)
                errexit("awareClient: can't open TCP socket %d\n",WSAGetLastError());

        /*
         * Bind any local address for us.
         */

        memset(&awareClient, 0, sizeof(awareClient));       /* zero out */
        awareClient.sin_family      = AF_INET;
        awareClient.sin_addr.s_addr = htonl(INADDR_ANY);
        awareClient.sin_port        = htons(0);
        if (bind(awareSocket, (struct sockaddr *) &awareClient, sizeof(awareClient)) ==
SOCKET_ERROR)
                errexit("client: can't bind local address: %d\n",WSAGetLastError());
```

```
   /* Connect the socket */
   if (connect(awareSocket, (struct sockaddr *)&awareServer, sizeof(awareServer)) ==
SOCKET_ERROR)
        errexit("awareClient can't connect to awareServer : %d\n", WSAGetLastError());

        while(1)
        {

                awareMessage();
    //  getchar();
        }

}

/*************************************************************************
 awareMessage()
 *************************************************************************/
void awareMessage(void)
{
        int     n;              /*  read count */

        FILE    *stream;                        /* */
        static char     line;                   /* */  // Important: initialize to 0
   //   int             outchars, inchars;      /* */

        messageBuf.xPos=0;
        messageBuf.yPos=0;

        tempBuf.xPos = htons(messageBuf.xPos);
        tempBuf.yPos = htons(messageBuf.yPos);


        (void) send(awareSocket, (char *)&tempBuf, sizeof tempBuf,0);

    /* read it back */
   //    for (inchars = 0; inchars < outchars; inchars += n)             /* */
   //      {
                                /* */
                n = recv(awareSocket, &line, 2,0);                                      /* */
   //       n = recv(awareSocket, &line[inchars], outchars-inchars,0);          /* */
   //  n = recv(awareSocket, (char *)&tempBuf, sizeof tempBuf,0);

       if (n  == SOCKET_ERROR)
          errexit("awareClient: socket read failed: %d\n", WSAGetLastError());
  //       } // end for
                                /* */
```

```
        messageBuf.xPos=ntohs(tempBuf.xPos);
        messageBuf.yPos=ntohs(tempBuf.yPos);

        stream = fopen("myvar.xyz", "w");    /* */
//      fprintf(stdout, "%c", line);
        fputs((&line), stream);                                  /* */
//      fprintf(stdout,              "received              xPos:              %d\tyPos:
%d\n",messageBuf.xPos,messageBuf.yPos);
        fclose(stream);
        Sleep(500);                  // Allow enough time for file I/O
}

/*********************************************************************
 errexit()      :      Print error message and exit
 *********************************************************************/
void errexit(const char *format, ...)
{
        va_list args;

        va_start(args, format);
        vfprintf(stderr, format, args);
        va_end(args);
        WSACleanup();
        exit(1);
}
```

Appendix C

Multimedia Authoring Tools

and

Authorware

C.1 What is a multimedia-authoring tool?

Multimedia authoring tools are the software programs used to effectively orchestrate all different building blocks of multimedia to create a powerful, synergistic final product. Most authoring tools allow the developer to create and modify backgrounds and visual effects for a project's interface. Many of them allow the author to control external devices such as videotape decks and laserdisc players. More importantly, such tools can also create links between graphics, text, and audiovisual elements.

C.2 Evaluation and selection of the authoring software

Usually, when evaluating authoring software, issues such as interface, capabilities, and features are carefully considered. In this research project, many of the existing multimedia authoring tools were reviewed. A few significant names among them include:

- Macromedia Authorware Attain
- Macromedia Director
- Apple Media Tool
- Asymetrix Toolbook

Apart from the above-mentioned issues, a few others were also needed to be considered before 'Macromedia Authorware' was selected for this research project. Authoware is a cross-platform development tool that can be used to create variety of multimedia production. It mainly focuses on training applications. More importantly, Authorware's capacity for the desired network functionality was studied and greatly appreciated. Authorware was selected as the tool to be used for the proposed system development. Following sections compare between two prominent authoring tools in the market, viz.

Authorware and Director and also take a closer look at Authorware's capabilities and important features. These sections explain why Authorware was made the choice for this research project.

Differences Between Authorware and Director

- Macromedia Director

Macromedia Director is one of the most popular authoring tools. Director's strengths include the ability to create animations and control such external devices as laserdiscs, and its custom scripting environment. For many projects that include 2-D animation, sound, and interactivity, almost all of the work can be done within Director itself.

- Macromedia Authorware

Macromedia Authorware is one of the leaders for interactive authoring. Authorware is a cross-platform development tool that can be used to create any type of multimedia production, but focuses on training applications. Its greatest strength is that it is entirely based on using icons for creating interactive links; so there is no scripting necessary.

Both programs can do many of the same things but employ different metaphors (and Graphical User Interfaces, GUIs) to accomplish these tasks. While Director is the more efficient program for controlling animation and sound, Authorware is better for setting up interactions that track what a user does, such as checking how someone assembles objects on the screen or recording a number of times someone performs a task.

Director's strong points include :

- Animation

  Director is noted for its ability to create animation efficiently and precisely.

- Sound

  Director can play two simultaneous sounds on Windows computers and eight simultaneous sound on Macintosh computers. When you play sound from Director, you have more control over sound volume and over sound start and stop points.

- Synchronized sound and graphics

  Director's precise control of sound and graphics lets the developer synchronize the two more exactly.

- Control through Lingo

  With the Lingo scripting language you can control such objects as sprites (sprites are programmable stage characters on a development platform), sound, and other movie components. For example, Lingo can change sprite size and location in response to a user action or create a series of additional objects through the use of parent scripts.

Authorware's strong points include :

- Navigation

  Authorware provides powerful tools to move a Director animation around the screen.

- Setup of interactions

  Authorware's flowline structure and interactions are suited to setting up a variety of interactions quickly.

- Ordering objects and events

  Authorware's flowline lets you quickly determine when objects appear-- The flowline also give you a visual representation of how the icons relate to each other.

- Tracking user actions

Authorware provides many variables that record what the user does. When a Director movie plays within an Authorware piece, the user can interact with both of them.

C.3 Authorware Internal System Functions

System variables in Authorware software program are used to store pieces of information. Authorware functions are used to perform special tasks. The Authorware calculation icon is typically where Authorware scripting takes place. The calculation icon is a container for expressions, variables, and functions. How Authorware System functions are organized : Authorware system functions have been separated into fourteen categories and are listed alphabetically within those categories.

- Character

These Authorware functions are used when dealing with text and strings. For example, you can use these functions to count the number of characters in a string, get a specific line of text from a string, or find a pattern of characters in a string.

- File

These functions enable you to create and work with external files. For example, you can create a directory, write a text file in that directory, then catalogue all of the files within that directory

- Framework

These functions enable you to work within a framework. For example, you can conduct a search for content text or keywords, you can get a list of key words for a certain icon, or create a list of the past pages visited by the end user.

- General

The General category functions often perform general system level tasks. For example, you can press a key, turn off the cursor, or copy text to the Clipboard.

- Graphics

These functions affect how graphics appear in the Presentation window. For example, you can set a fill pattern, determine the RGB color, or draw an object, such as a line or a box.

- Icons

These functions manage icons from behind the scenes. For example, you can display and erase icons, change the display layer for the objects in an icon, or get the title of a particular icon.

- Jump

The Jump functions enable the piece to leap from one icon to another as well as jump out to external files. For example, you can set Authorware to launch other applications, jump to a specific icon when a determined amount of time passes without any user activity, or to jump from one Authorware piece to another.

- Language

These functions perform specific programming operations, such as If-Then statements, Repeat While loops, and assigning a value to a variable.

- Math

These Authorware functions perform math operations. From simple additions to determining the cosine of an angle, Authorware can perform most needed math functions.

- OLE (Object Linking and Embedding)

The OLE functions are used to handle OLE objects within Windows. All standard OLE communications are supported by Authorware.

- Platform

The platform functions are typically used to get information that will later be used by XCMDs (stands for Xcommands on a Macintosh platform) or DLLs (stands for Dynamic Linked Libraries on a windows platform). For example, one Platform function will return the creator type to determine if Authorware is currently running application.

- Time

These functions convert aspects of time and date to a numerical format so that they can be used to make comparisons. For example, one function converts today's date to a number representing the number of day's since January1, 1990.

- Video

The Video functions are used to control the playing of video through video overlay cards via a laserdisc player. The functions seek out specific frames, pause the video, or set the chroma key value.

- Network

The network functions are used for pieces that will be run over the Internet. Accessing external data or downloading content from the Web to a local hard drive is accomplished through such functions.

In addition to the fourteen categories of functions listed in the Category pop-up menu, there are two other selections:

- All

This option lists all of the Authorware functions in alphabetical order.

- Custom

The list corresponding to custom functions contains all of the functions that one has created and imported using DLLs (Dynamic Linked Libraries for Windows platform) or XCMDs (X-commands for Macintosh platform).

Appendix D

TCP/IP Terminology

and

Client-Server Principle

TCP/IP Software Design Issues : [11]

From the viewpoint of an application, TCP/IP, like most computer communication protocols provides basic mechanisms used to transfer data. In particular, TCP/IP allows a programmer to establish a communication between two application programs and to pass data back and forth. Thus, TCP/IP provides peer-to-peer communication. The peer applications can execute on the same machine or on different machines. The method is known as the client-server paradigm. In fact, client-server interaction has become so fundamental in peer-to-peer networking systems that it forms the basis for most computer communication.

The fundamental motivation for the client-server paradigm arises from the problem of rendezvous. The client-server model solves the rendezvous problem by asserting that in any pair of communicating applications, one side must start execution and wait (indefinitely) for the other side to contact it. The solution is important because TCP/IP does not respond to incoming communication requests on its own. Because TCP/IP does not provide any mechanisms that automatically create running programs when a message arrives, a program must be waiting to accept communication before any requests arrive. Thus, to ensure that computers are ready to communicate, most system administrators arrange to have communication programs start automatically whenever the operating system boots. Each program runs forever, waiting for the next request to arrive for the service it offers.

Terminology and Concepts :

The client-server paradigm divides communicating applications into two broad categories, depending upon whether the application waits for communication or initiates it. In

general, an application that initiates peer-to-peer communication is called a client. End users usually invoke client software when they use a network service. Most client software consists of conventional application programs. Each time a client application executes, it contacts a server, sends a request, and awaits a response. When the response arrives, the client continues processing. By comparison, a server is any program that waits for incoming communication requests from a client. The server receives a client's request, performs the necessary computation, and returns the result to the client.

Connectionless Vs. Connection-Oriented Servers :

When programmers design client-server software, they must choose between two types of interaction: a connectionless style or a connection-oriented style. The two styles of interaction correspond directly to the two major transport protocols that the TCP/IP protocol suite supplies. If the client-server communicate using UDP, the interaction is connectionless; if they use TCP, the interaction is connection-oriented.

From the application programmer's point of view, the distinction between connectionless and connection-oriented interactions is critical because it determines the level of reliability that the underlying system provides. TCP provides all the reliability needed to communicate across an internet. It verifies that data arrives, and automatically retransmits segments that do not. It computes a checksum over the data to guarantee that it is not corrupted during transmission. It uses sequence numbers to ensure that the data arrives in order, and automatically eliminates duplicate packets. It provides flow control to ensure that the sender does not transmit faster than the receiver can consume it. Finally, TCP informs both the client and server if the underlying network becomes inoperable, for any

reason. By contrast, clients and servers that use UDP do not have any guarantees about reliable delivery. When a client sends requests, the requests may be lost, duplicated, delayed, or delivered out of order. Similarly, the responses the server sends back to a client may be lost, duplicated, delayed, or delivered out of order. The client and/or server application programs must take appropriate actions to detect and correct such errors.

Most experienced professionals, prefer to use the connection-oriented style of interaction. A connection-oriented protocol makes programming simpler, and relieves the programmer the responsibility to detect and correct errors.

In nutshell, when designing client-server applications, programmers are strongly advised to use TCP because it provides reliable, connection-oriented communication. Programs only use UDP if the application protocol handles reliability, the application requires hardware broadcast or multicast, or the application cannot tolerate virtual circuit overhead.

Concurrent Processing in Client-Server Software :

The term concurrency refers to real or apparent simultaneous computing. For example, a multi-user computer system can achieve concurrency by time-sharing, a design that arranges to switch a single processor among multiple computations quickly enough to give the appearance of simultaneous progress, or by multiprocessing, a design in which multiple processors perform multiple computations simultaneously.

Concurrent processing is fundamental to distributed computing and occurs in many forms. Among machines on a single network, many pairs of application programs can communicate concurrently, sharing the network that interconnects them. For example, application A on one machine may communicate with application B on another machine,

while application C on a third machine communicates with application D on a fourth. Although they all share a single network, the applications appear to proceed as if they operate independently. The network hardware enforces access rules that allow each pair of communicating machines to exchange messages. The access rules prevent a given pair of applications from excluding others by consuming all the network bandwidth.

Concurrency can also occur within a given computer system. For example, multiple users on a timesharing system can each invoke a client application that communicates with an application on another machine. One user can transfer a file, while another user conducts a remote login session. From a user's point of view, it appears that all client programs proceed simultaneously. In addition to concurrency among clients on a single machine, the set of all clients on a set of machines can execute concurrently.

In contrast to concurrent client software, concurrency within a server requires considerable effort.

Program Interface To Protocols :

This section describes general properties of the interface an application program uses to communicate in the client-server model. A program interface must support the following conceptual operations:

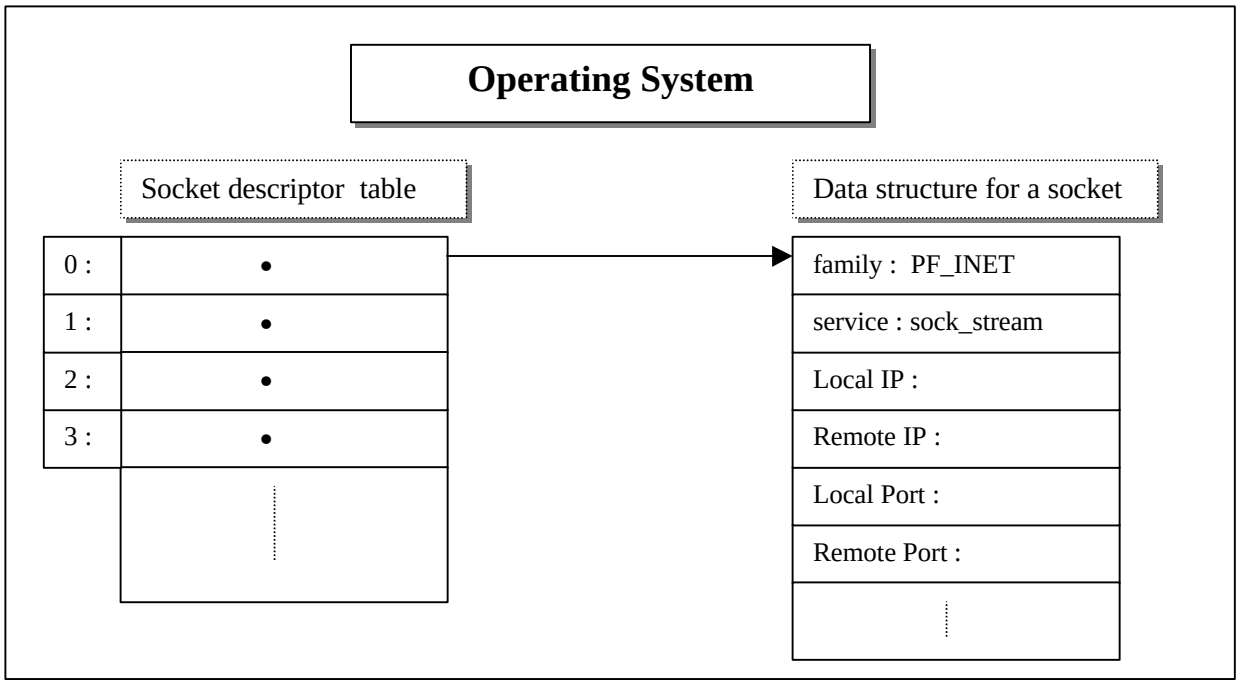
- Allocate local resources for communication
- Specify local and remote communication endpoints
- Initiate a connection (client side)
- Wait for an incoming connection (server side)
- Determine when data arrives

- Generate urgent data
- Handle incoming urgent data
- Terminate a connection gracefully
- Handle connection termination from the remote site
- Abort communication
- Handle error conditions or a connection abort
- Release local resources when communication finishes

Introduction to Socket API

The socket abstraction was introduced by the BSD UNIX operating system as a mechanism that allows application programs to interface with protocol software. Because many vendors have adopted sockets, they have become a de facto standard. The socket interface adds a new conception for network communication, the socket. Like files, each active socket is identified by an integer called its socket descriptor. The Windows operating system keeps a separate table of socket descriptors for each process.

The socket API contains a function, socket, that an application calls to create a socket. When an application calls socket, the operating system allocates a new data structure to hold the information needed for communication, and fills in a new entry in the process' socket descriptor table with a pointer to the data structure. The following figure illustrates a process' socket descriptor table after a call to socket.



Once a socket has been created, it can be used to wait for an incoming connection or to initiate a connection. a socket used by a server to wait for an incoming connection is called a passive socket, while a socket used by a client to initiate a connection is called an active socket. The only difference between active and passive sockets lies in how applications use them; the sockets are created the same way initially.

Additional system calls allow the application to specify a local endpoint address (bind), to force the socket into passive mode for use by a server (listen), or to force the socket into active mode for use by a client (connect). Servers can make further calls to obtain incoming connection requests (accept), and both clients and servers can send or receive information (recv and send). Finally, clients and servers can deallocate a socket once they have finished using it (closesocket).