Kumaraguru College of Technology

Department of Computer Science and Engineering
Coimbatore-641 006

ISO 9001:200

P-1096

MINISERVER

Project work done at

SRM SYSTEMS AND SOFTWARE LIMITED

PROJECT REPORT

Submitted in partial fulfillment of the

Requirements for the award of the degree of

Master of Science in Applied Science

Software Engineering

Bharathiar University, Coimbatore

Submitted by

R.RAMYA
Reg.No-0037S0098

INTERNAL GUIDE

Mr.G.S Nanda kumar.B.E.,
Dept.of Computer Science& Engineering,
Kumaraguru College of Technology,
Coimbatore.

EXTERNAL GUIDE

Mr.S.Kamesh B.E,
SRM SYSTEMS AND SOFTWARE. Ltd,
CHENNAI

Department of Computer Science and Engineering

KUMARAGURU COLLEGE OF TECHNOLOGY

Coimbatore – 641 006

ISO 9001:200

CERTIFICATE

PROJECT REPORT 2003

Certified that this is a bonafide report of
the project work done by

R.RAMYA
(Reg. No. 0037S0098)

G.S Nanda Kumar 27/09/03

Mr.G.S Nanda Kumar.B.E.
Project guide
Computer Science & Engineering

S. Thangasamy

Prof. S. Thangasamy , Ph.D.,
Head of the Department
Computer Science & Engineering

Place: Coimbatore
Date:

Submitted for University examination held on

29/9/03

Internal Examiner

S. Thangasamy 29/9

External Examiner

29/9

SRM SYSTEMS AND SOFTWARE LIMITED
24, G.N. Chetty Road, T.Nagar, Chennai - 600 017.
✆ : 91 - 44 - 8250771, 8258757, 8269471 Fax : 91 - 44 - 8283359
E-mail : srm@srmsoft.co.in Web Site : http://www.srmsoft.com
Regd. Off : 2, Veerasamy St., West Mambalam, Chennai - 600 033.

SRM SYSTEMS AND SOFTWARE

23.09.2003

CERTIFICATE

This is to certify that the project work entitled "**MINI SERVER**" was Analyzed, Designed and Developed by **Ms. R.RAMYA** of **KUMARAGURU COLLEGE OF TECHNOLOGY (CBE)**, submitted in partial fulfillment of the requirements of degree of **4th Year M.Sc (S.E)** has been carried out in our organization from June 2003 to Sep 2003.This project has been developed using VC++.

We wish him success in all his future endeavors.

For SRM Systems And Software Limited

Manager-Projects

DECLARATION

I here by declare that the project entitled **"MINISERVER",** submitted to **Kumaraguru College of Technology**, Coimbatore Affiliated to Bharathiar university as the project work of **Master of Science in Applied Science Software Engineering** ,is a record of original work done by me under the supervision and guidance of **Mr.S.Kamesh.B.E,** SRM Systems And Software Ltd., Chennai and **Mr.G.S.Nanda kumar.,B.E.,** CSE Department Kumaraguru College of Technology, Coimbatore and the project work has not found the basis for the award of any Degree/Diploma/Associateship/Fellowship or similar title to any candidate of any University.

Place: Coimbatore

Date: 27/9/03

R. Ramya

(R.RAMYA)

Reg.No:0037S0098

Countersigned by

Sundaram 27/9/03

(Internal Guide)

Mr.G.S.Nanda Kumar, B.E.,

Kumaraguru College of Technology,

Coimbatore.

ACKNOWLEDGEMENT

I am immensely grateful to **Dr.K.K.Padmanaban BSc(Engg) , M.Tech., Ph.D.,** Principal , Kumaraguru College of Technology for his valuable support to come out with this project.

I really feel delighted in expressing my heartful thanks to **Dr.S.Thangaswamy Ph.D**, Prof & Head of Department of Computer Science and Engineering for his endless encouragement in carrying out this project successfully.

My heartfelt thanks to our project coordinator **Mrs.S.Devaki B.E., M.S,** Assistant Professor, for his unfailing enthusiasm, encouragement and guidance that paved me to the completion of this project.

I am indent to express my heartiest thanks to **Mr.G.S Nanda Kumar** my project guide who rendered his valuable guidance and support to do this project work extremely well.

I am greatly indebted to chairman SRM Systems And Software Limited, chennai, for getting us into his esteemed institution. I also thank **Mr.S.Kamesh** B.E who was my guide and he has helped me a lot in my project.

I am also thankful to all the faculty members of the Department of Computer Science and Engineering, Kumaraguru College of Technology, Coimbatore for their valuable guidance, support and encouragement during the course of my project work..

My humble gratitude and thanks to my parents who have supported, to complete the project and to my friends, for lending me valuable tips, support and cooperation through out my project work.

SYNOPSIS

SYNOPSIS

The project titled "Miniserver" is a system side project for the concern SRM Radiant Infotech Ltd. using vc++ platform. The concern has the capability to provide complete end to end IT solutions to its customers through its matchless portfolio of products,services and a large infrastructure network.

The project is to implement a server that gives users access to operating system functions and programs over the Internet through a standard browser. Much of the functionality of a Web server will be implemented, including the ability to execute CGI (Common Gateway Interface) programs. Commands or programs can be entered from a browser, sent to the server, executed, and the program output displayed back to the browser. From a browser, it will be possible to control a remote computer's operation over an Internet connection.

The ***remote computer*** **or** ***server*** will execute C++, Assembly programs and many DOS commands from a browser, providing access and control of the server computer from across the Internet.

The server recognizes a limited number of commands that are implemented internal to the server program as regular C++ or Assembler functions. Any command not recognized internally is passed on to DOS which attempts to execute the command as a program contained in the server directory or as a DOS command. Essentially any well behaved program that can be executed from the DOS command line can also be executed by the server.

The program uses standard input and standard output.

CONTENTS PAGE NO

1. INTRODUCTION

1.1 Current Status of the Problem:

Computer operating systems provide management and access services for the hardware resources of the system. These services are often accessible at the programmer level as callable functions or at the user level through some interface such as a command line in DOS and Unix or a GUI as in Mac or Windows.. In DOS one types *dir* and the *command.com* program reads the command and executes the *directory* listing program or function. In Windows one clicks the Windows Explorer icon to perform much the same operation. This requires that every system to have DOS installed in it and the user does not have the facility of remote accessing or remote manipulation . The user on net has to open the DOS window each time to know the files present etc... rather than to find out directly from the browser. Moreover DOS only has limited commands and does not facilitate user defined commands.

1.2 RELEVANCE AND IMPORTANCE

1.2.1 Existing System

The existing system does not allow remote accessing or remote manipulation. The drawbacks of the system are

1. The software needs to be installed in each system
2. User defined commands cannot be added
3. DOS commands could not be executed via browser.

1.3 Need for Proposed System

Owing to the above mentioned drawbacks in the existing system, an automated system is proposed. The proposed system aims to eliminate these drawbacks. It can be viewed as user friendly, efficient easing the work of the user

The benefits of the proposed system are

- User defined commands can be added.
- DOS commands can be executed via browser.

1.4 Proposed System:

The proposed system is aimed at providing remote access and installation and eases the task of a user. The proposed system is user friendly.

The proposed system has been developed using VC++ overriding its foundation classes. Since it makes use of TCP/IP protocol the proposed system will overcome the drawbacks of the existing system. The user defined commands are developed in assembly language.

The advantages of the proposed system are

- Remote access and installation.
- Increases the speed.
- Decreases the memory.
- Increases the scope of the browser.

2. COMPANY PROFILE

SRM SYSTEMS AND SOFTWARE is a company committed to provide support to small, medium and large corporations in the development and management of software essential to their needs over the entire life cycle of a project or system. All corporations, regardless of size, need to process enormous amounts of data in support of the day-to-day operation of the company and the dependence on a corporate information system and up-grade the existing ones. In seeking efficient and cost-effective approaches to manage change, many companies have found outsourcing to be particularly attractive.

SRM Systems and Software is here to provide expert services and support for "change management" in software systems allowing your organization to focus on its core business. SRM Systems and Software offers the expertise of experienced individual software consultants, as well as an offshore facility with a state-of-the-art information technology infrastructure and a well-trained and committed staff, all at extremely competitive prices. We at SRM provide our clients the potential for significant savings without a compromise in quality or schedule. SRM Systems and Software guarantees that the software services will be delivered to the customer on time, within budge, incomplete conformance means that at SRM, we are indeed
" Determined to Make a Difference".

MISSION STATEMENT

The stated mission of the SRM System and Software is to offer value addition to the customer's Business through IT Solutions of high quality and appropriate Technology on time and on budget.

CORPORATE BACKGROUND

- Reputation built over 3 decades
- Global vision
- Asset base of over US $100 million

- Many interests but one objective - Commitment to Excellence

SRM Systems and Software is a unit of the renowned SRM Group, which in the past30 years has established itself in Southern India in the field of Engineering education and Research. Over the years, the SRM Group, with an asset base of more than US $ 50 million, has expanded into the fields of Health Care, Hospitality, Manufacturing, Financial Services and Construction.

SRM Systems and Software was established with a specific business focus on Software Development and consultancy. As a member of the Software Technology Park of India, SRM Systems and Software benefits through business and customs duty incentives from the Government of India and consequently is committed to export 100% of its products and services.

The overseas office of SRM Systems and Software in Boston provides an effective link to customers in the United States and other parts of the world. Efforts are under way to establish similar offices in Japan, UK, Europe and Australia. Connected by broadband data links, the Headquarter in Chennai and the overseas offices will be positioned to provide customers global information technology market by an unwavering commitment to quality.

SRM Systems and Software - A Customer Centric Company

OBJECTIVES

- World Class Products
- Commitment to Quality
- Impeccable Customer Service
- Excellent Technical Support

BUSINESS ETHICS

- Customer is God
- Work is Worship
- Employee is Strength
- Humanity is the Base

STRATEGY

- Our International strategy is to penetrate and service the market by On-site, Off shore & Turnkey projects based on our expertise and related software solutions
- Our Domestic Strategy in India is to increase market share, expand Client base and focus on large IT contracts.

UNIFIED STRENGTH

- Three decades of SRM's Track Record
- Strong Team Work
- Excellent Technical Competence
- Structured Project Approach
- Customer Centric and Focus on Customers' Customers
- Japanese Language Competence

SERVICES OFFERED

SRM Systems and Software through its Strategic Business Units offers the following services.

CUSTOMIZED SOFTWARE DEVELOPMENT

SRM can provide complete business turnkey solutions to small, medium and large size companies spanning every phase of the software life cycle: System Analysis, Design, Implementation, Testing, Installation and Maintenance. The SRM staff has an

accumulated experience of more than 300 man-years in varied application areas. SRM offers software services in the following technology areas:

- Web Based Applications and e-commerce
- Client-Server (two-three and n-tier Technology)
- Group Ware and Workflow
- Multimedia and Computer Graphics
- Computer Aided Design and Computer Aided Manufacturing

SRM guarantees each customer that any project executed by SRM will be developed as per the specifications, delivered on time, and without cost overrun. SRM strictly adheres to the latest Software Engineering standards in the development of customized software. The aim of SRM is to win the allegiance of each customer so that the relationship does not end with the completion of the first contract but becomes an ongoing and mutually beneficial association.

3. HARDWARE AND SOFTWARE SPECIFICATION

3.1 Hardware Specification

System	Pentium III @600MHZ
Cache	128 MB
RAM	128 MB
Hard Disk	20 GB
Monitor	14" Color Monitor
Keyboard	104 Enhanced
Mouse	Logitech three button mouse
NIC	LAN cord

3.2 SOFTWARE SPECIFICATION

Operating system: Windows workstation

Software required:

- Visual C++
- MS-DOS
- Assembly Level Language

PROPOSED APPROACH TO THE PROJECT

4. PROPOSED APPROACH TO A PRODUCT

Software used to develop the system is Visual C++,

4.1 VISUAL C++:

Visual C++ has various features for which it is selected. It has very good compiling tools. Some of the features of Visual C++ are

- Supports network communication programs
- Supports ActiveX, ODBC, OLE
- Easy to handle graphics and animation
- Easy to write threading applications

4.2 ASSEMBLY LEVEL LANGUAGE

- A *low-level processor-specific* programming language design to match the processor's machine instruction set
- Each assembly language instruction matches exactly one machine language instruction
- we study here Intel's 80x86 (and Pentiums)

4.2.1 Features of Assembly Level Language

- To learn how high-level language code gets translated into machine language
  - i.e.: to learn the details hidden in HLL code
- To learn the computer's hardware
  - by direct access to memory, video controller, sound card, keyboard...
- To speed up applications
  - direct access to hardware (ex: writing directly to I/O ports instead of doing a system call)

– good ASM code is faster and smaller: rewrite in ASM the critical areas of code

4.2.2 Usage of Assembly Language

There is some debate over the usefulness of assembly language. In many cases, modern compilers can render higher-level languages into code as that runs as fast as hand-written assembler.

However, some discrete calculations can still be rendered into faster running code in assembler, and some low-level programming is simply easier to do in assembler. Some system-dependent tasks performed by operating systems simply cannot be expressed in high-level languages. Many compilers also render high-level languages into assembler first before fully compiling, allowing the assembler code to be viewed for debugging and optimization purposes.

Many embedded devices are also programmed in assembly to squeeze the absolute maximum functionality out of what is often very limited computational resources, though this is gradually changing in some areas as more powerful chips become available for the same minimal cost.

DETAILS OF THE DESIGN

5.DETAILS OF THE DESIGN

5.1 SYSTEM DESIGN

The system design phase follows system analysis. It provides the information that is to be fed in the system and how the output is obtained. The design goes through logical and physical stages of development. Logical design the physical system, prepares input and output specification,makes the browser send and receive input and output from the remote server.The physical design maps out the details of the physical system,plans the system implementation.

5.2 INPUT DESIGN AND OUTPUT DESIGN:

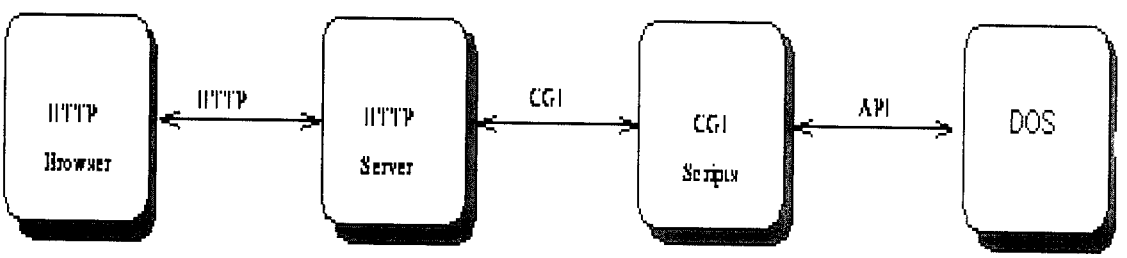
The definition of a well behaved program is one that uses standard input and standard output. The basic idea is that a program that reads standard input and writes standard output can read the keyboard or a file, write the screen or a file, without any changes to the program. This is performed by the operating system through redirection. For example, the following C++ program reads until a ' ' or blank is entered, converting characters 'A'-'Z' to lowercase and outputting (note that other characters might be mangled).

Suppose the name of the program executable is lowcase.exe. Once compiled, the program can be executed several ways.

- lowcase - The input is from the keyboard and output to the screen.
- lowcase < a2z - The input is from the file a2z and output to the screen.
- lowcase > z2a - The input is from the keyboard and output to the file z2a.
- lowcase < a2z > z2a - The input is from the file a2z and output to the file z2a. Assuming that the a2z file contained the one line:
  ABCDE Z 12345
  the program would output to the file z2a the characters through the first ' ' converted to lowercase as: abcde
- lowcase ABCDE Z 12345 - From the browser as in the second figure below. The third figure is the resulting output, stopping at the first ' '.

5.3 SYSTEM FLOW DIAGRAM

The system flow diagram gives a clear picture of a system, which is a network of functional processes connected to one another by data. System flow diagrams is particularly used for operational systems in which the functions of the system are important. System flow diagrams are used in software engineering field as a notation for studying system design issues.This provides the functional view of the system.



The input string fed to the browser is treated as HTML format. The format is then sent to the server through HTTP. The common gateway interface is used for converting the HTML format .The CGI produces the script for the given HTML input which is filtered and transferred to DOS through API.

The output obtained undergoes similar conversion from the script to the HTML format which is displayed on the client browser.

5.4 NETWORK DIAGRAM

The diagram shows the client-server nature of the miniserver. The communication is done over LAN network via browser.

PORTS:

A port is a special memory location that exists when two computer are in communication via TCP/IP. Application uses a port number to an identifier to other computers. Both sending and receiving computers use this port to exchange data.

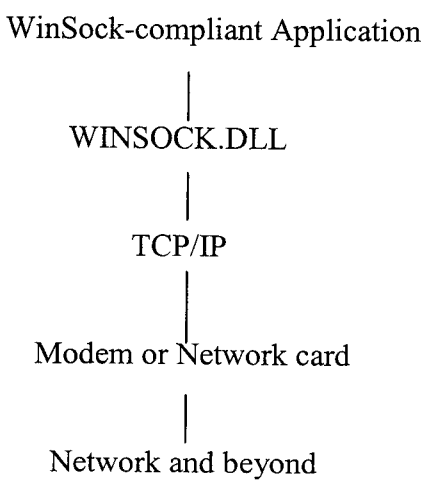
SOCKET:

Combination of an IP address and port number. It provides point to point, two way communication between the processes.

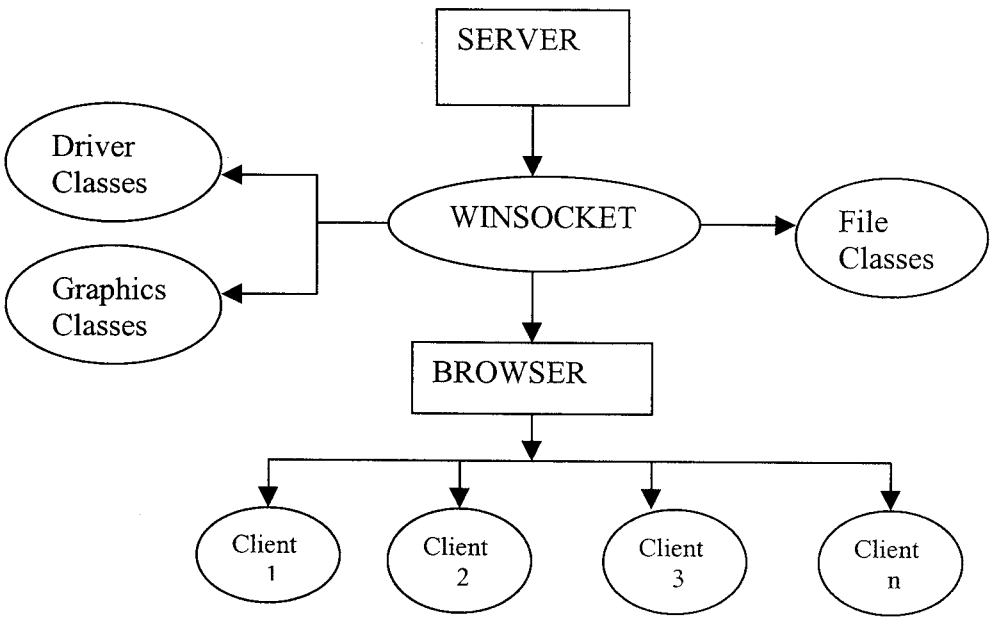
WINSOCK:

Winsock is a DLL (Dynamic Link Library) and runs under Windows 3.x, Windows for WORKGROUPS, WINDOWS NT, AND WINDOWS 95. The WINSOCK.DLL is the interface to TCP/IP and, from there, on out of the internet.

The diagram shows how it works:



WINSOCK.DLL actually acts as a "layer" between your Winsock application and TCP/IP stack.



5.5 CODE DESIGN:

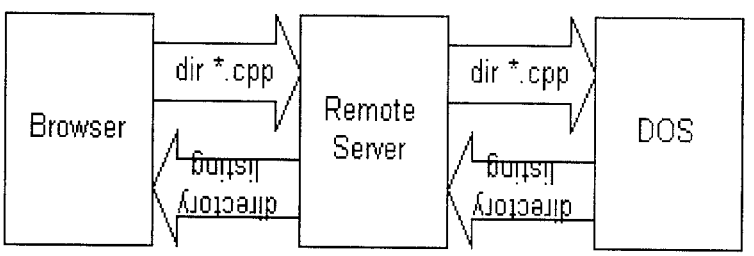
In this design an object physical characteristics or performance characteristics or operational instructions are specified. This can also show the inter relationship and may sometimes be used to achieve secrecy or confidentiality.

The development methodology is used in the code design. The approach used here is the top-down approach. Here codes are used for capturing, sending files and signals across networks.

5. 6 PROJECT DESCRIPTION

A minimal server has been implemented in a combination of C++ and Assembler as a starting point. The main project tasks are to: a) translate several of the existing C++ functions into Assembler, b) extend the server to perform additional functions, and c) implement some useful CGI operations as Assembler programs.

The following diagram illustrates how the browser would get a directory listing of the server computer. The user typed *dir* into the browser, which sends the request to the server, which executes the DOS *dir* command. The



dir command prints the directory listing to standard output, which is sent back to the server, and finally back to the browser, where it is displayed.

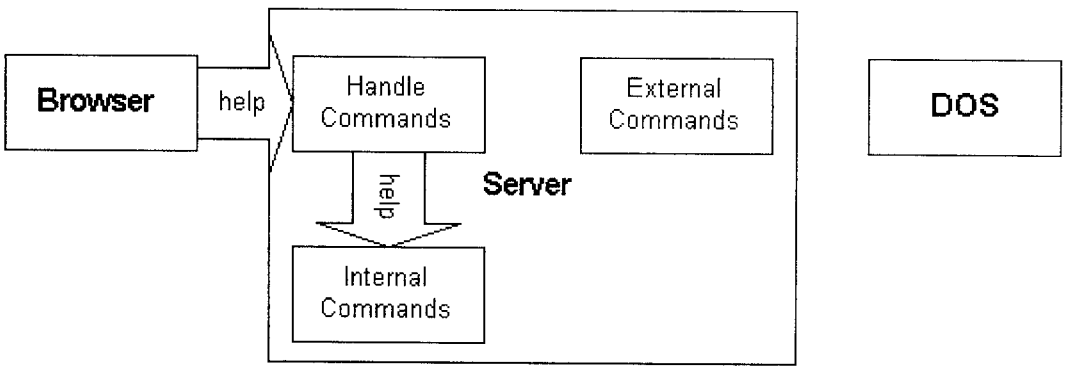
The following three figures depict:

- User input of *dir *.cpp* to display the directory of only the C++ files.
- The directory results displayed by the browser.
- The server trace window showing the input to the server from the browser and the output from the *dir* execution that was sent o the browser.

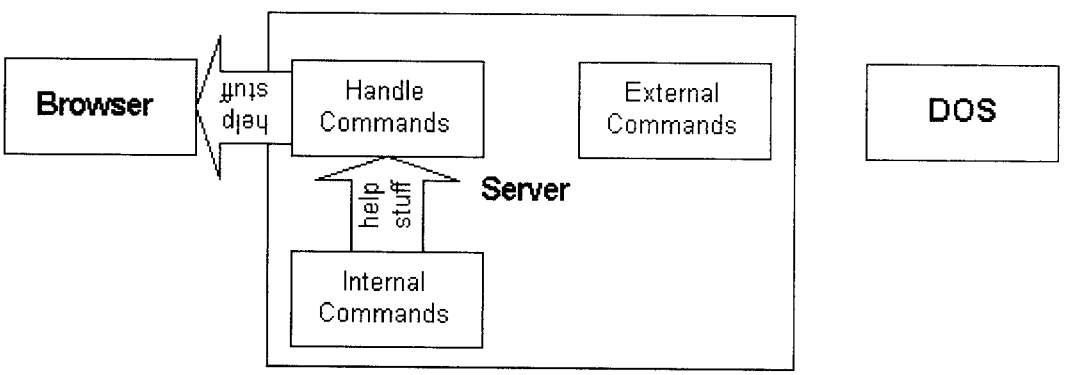
5.6.1 How the Server Works

The server talks to the browser. Everything the server inputs from the browser, works with, and sends back to the browser is a *string*.

Browser sends user input "*Help*" to server. Server recognizes "Help" as an internal server command.



Server responds by sending HTML text for the "Help" command.

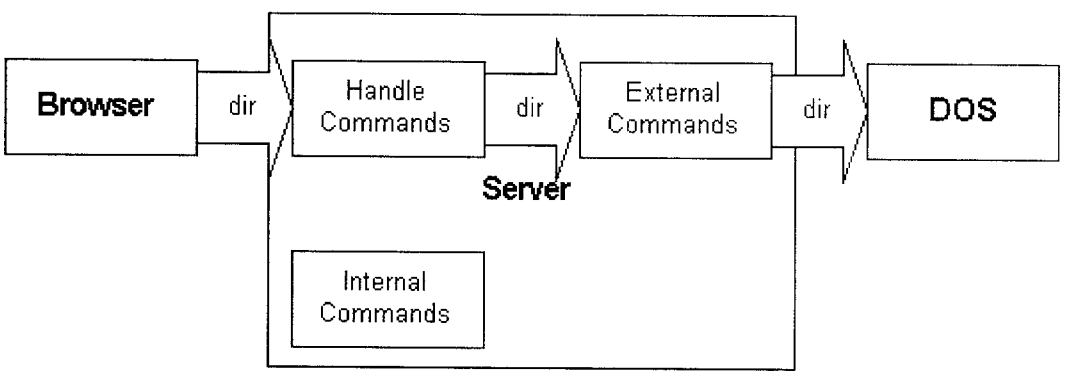


The server, as illustrated above, gets its input from the browser. This input is just the string that the person using the browser typed in before hitting the Enter key. The server examines this input to determine whether either an *internal* command, one that the server can handle itself, or an *external* command, one that the server doesn't have a clue about and passes on. The server examines the input from the browser and determines whether the first word the user typed was a server command. Basically, the command tells the server what internal server function the user wants executed. If the first word is an internal server command, the server just executes the appropriate C++ or Assembler function. In the above figure the input is a server command *help* which is handled

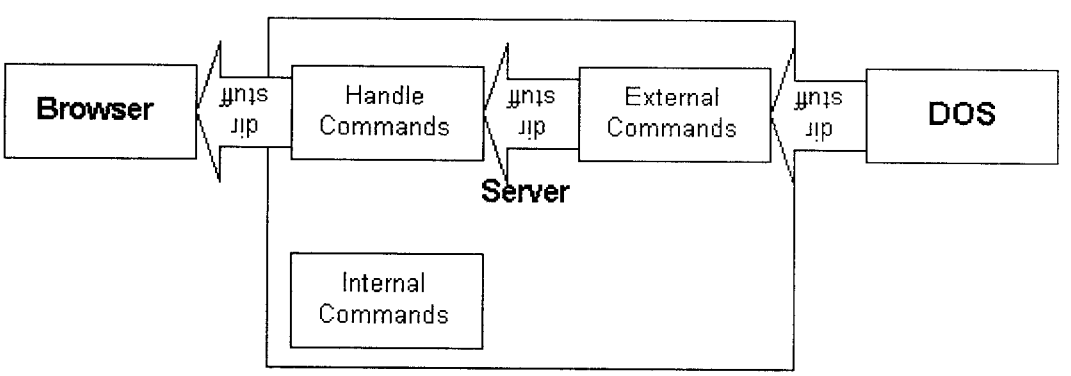
internally by the server executing the HELP function and sending back a string of *help stuff*.

If the server doesn't recognize that first word as a command, the server lets DOS have a go at executing that word as the name of a DOS program. In the below diagrams, the *dir* command is unrecognized by the server so is passed on to DOS which returns a string containing the server directory listing.

Browser sends user input "*dir*" to server. Server fails to recognize "dir" as an internal server command and passes "dir" text to DOS.



DOS executes "dir" returning generated text back to server. Server returns text From DOS to browser.



The function *EXECUTEcmd* that distinguishes between *internal* server and *external* DOS commands is given below. It receives two strings, *command* is the command such as *dir, lowcase, echo,* etc. and *input* is the input to the command or program.

For example, if *help* were entered in the browser, *command="help"* and *input=""* that is the empty string. The function attempts to match *command* with all the internal commands ("HELP", "STOP", and "VERSION") so would call the function HELP and return the output of the HELP function. If *echo hello* is entered at the browser then *command="echo"* and *input="hello"* and a call of *ECHO("hello")* is executed.

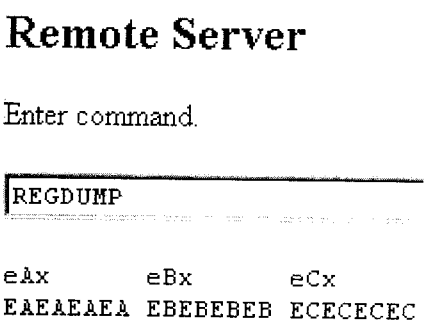
If *lowcase ABCD 1234* were entered in the browser, *command = "lowcase"* and *input="ABCD 1234".* None of the internal commands ("HELP", "STOP", and "VERSION") would match with *lowcase* so the command and input would be passed to the DOS function and the execution output returned. If *dir *.exe* is entered at the browser, since it matches neither *HELP, STOP, VERSION,* or *ECHO* a call of DOS("dir", "*.exe") is executed.

EXECUTEcmd - Handle Server and DOS commands

```
extern "C" char * EXECUTEcmd(char command[], char input[]) {
    if (STRcmp("HELP", STRupcase(command)) == 0)
        return HELP();
    if (STRcmp("STOP", STRupcase(command)) == 0)
        return STOP();
    if (STRcmp("VERSION", STRupcase(command)) == 0)
        return VERSION();
    if (STRcmp("ECHO", STRupcase(command)) == 0)
        return ECHO(input);
    return DOS(command, input);
}
```

5.6.2 Extending the Server

The server can be extended by adding new internal commands similar to HELP, "STOP", and "VERSION". The extensions could be original or expanding upon existing commands, for example the REGDUMP command which is already functional but only partially completed. The REGDUMP command displays the CPU eAx, eBx and eCx register values in hexadecimal, it can be extended to display additional registers, the stack, memory, etc. The REGDUMP command
display is as follows.



It is important to note that ALL commands must return a *pointer to a string*. The characters of the string are passed back by the server to the browser for display. In the How the Server Works section, the *dir* command returns a copy of a string from DOS, which the server returns back to the browser.

The method of extending the server is:

- Add the command to be recognized and the function call to the *EXECUTEcmd* function. To add the VERSION command enter the two lines of:
  if (STRcmp("VERSION", STRupcase(command)) == 0)
  return VERSION();
  which matches *command* enter from the browser with "VERSION" then calls the VERSION() function.

- Add the prototype of the function called when the command is entered, for the VERSION function:
  *extern "C" char *VERSION(void);*
- Write a C++ pseudocode then the assembler version in one file of the same name as the function. It is not required that function and file be the same but reduces confusion.
  - Use the same approach from Homework 8. There are several sample Assembler functions given that can be used as models, including the *VERSION.asm* file.
  - Assemble the function using (for VERSION.Asm): *tasm /mx /zi version*
  - Add the *object* and *assembler* files to the project.
  - Build the project.
  - Execute the server.

5.6.3 Converting Existing C++ Functions to Assembler

There are a number of C++ functions in file *student.cpp* that can be converted to Assembler. For example, the steps to converting STRlen to Assembler are:

- Delete all but the function prototype, leaving extern "C" int STRlen(const char s[]);
- Convert the C++ to Assembler.
- Add the *STRlen.obj* file to the project.
- Build and execute the server.
- Test

5.6.4 Adding DOS Commands

Writing a program that runs under DOS but is executed by the server offers the greatest opportunity for accessing hardware and using BIOS and DOS functions. The DOS commands can be any executable program that uses *standard input* and *standard output.*

Otherwise, the programs can be very 0similar to those implemented earlier in this course, except of course that *GetDec, PutDec,* etc. cannot be used for input and output. Normally, the program would be written entirely in Assembler, using the 16-bit register set and segment registers.

The output will be displayed in the browser just as it appears when executed at the DOS prompt. HTML tags can be added to the text to improve the appearance when displayed by the browser. For example, server output will be displayed as bold faced by the browser if the text is surrounded by and . For example, bold stuff will appear as **bold stuff.**

5.6.5 Design Constraints

The project is designed to be completed in several steps:

1. Start simple - The base project is written in C++ and some Assembler examples. Get this to work first by building assembling REGDUMP.asm by: *tasm /mx /zi REGDUMP*
    - Build the *Project.exe*
    - Execute to verify that the server works..
2. Remove one C++ function at a time and replace it with the corresponding Assembler function.
    - Delete everything from the first { to the last } of the function. For STRlen the changes would be:
        - extern "C" int STRLEN(char s[]) { int Cx=0; while (*s++ != '\0') Cx++; return Cx; }
        - extern "C" int STRLEN(char s[]);
    - Write the Assembler function.
        - STRlen proc Near

:

:

Ret

- Add Assembler object file to project.
- Assemble using *tasm /mz /zi filename*
- *Rebuild all* for *Project.exe* and execute.

3. Test project Assembler functions using same inputs as in Step 1. The results should be the same.

5.6.6 Suggested Extensions

Other possible extensions are below, are we can devise other, long dreamed of commands. These may be implemented either internally by adding to the server or externally as standalone DOS programs.

- DEL - Delete a file or files.
- SWAP - Swap contents of two files.
- TAIL - Display a file from tail to head.
- GREP - Scan a file for occurrences of a string and display lines on which they are found.
- TYPE - Display a file from head to tail.
- COPY - Copy file to another.
- APPEND - Append a file to another.
- PRINT - Print a file.
- DATE - Print and change system date.
- TIME - Print and change system time.
- DIR - File size, date, etc. to directory command.
- RESOURCES - Display available memory/disk used/available.
- REGDUMP - Extend the REGDUMP function for remaining registers including eBp and eSp before the call to REGDUMP.

IMPLEMENTATION DETAILS

6. IMPLEMENTATION DETAILS

A crucial phase in the system life cycle is the successful implementation of the new system design. Implementation is the stage of project when the theoretical design is turned into a working system. Implementation involves creating computer- compatible files, training the operating staff, and installing hardware, terminals and telecommunications network (where necessary) before the system is up and running. A crucial factor in conversion is not disrupting the functioning of the organization.

In system implementation, user training is crucial for minimizing resistance to change and giving the new system a chance to prove its worth. The training aids include user manuals, help screens, data dictionary, job aids etc..

There are three types of implementation:

- Implementation of a computer system to replace a manual system
- Implementation of a new computer system to replace existing system
- Implementation of a modified application to replace an existing one, using the same computer.

TESTING

7.SYSTEM TESTING

Testing is an activity to verify that correct system is being built and is performed with intent of finding faults in the system. Testing is an activity, however not restricted to being performed after the development phase is complete. But this is to be carried out in parallel with all stages of system development, starting with requirements specification. Testing results, once gathered and evaluated, provide a qualitative induction of software quality and reliability and serve as a basis for design modification if required. A project is set to the incomplete without project testing.

System testing is process of checking whether the development system is working according to the original objectives and requirements. The system should be tested experimentally with the test data so as to ensure that the system works according to the required specification. When the system is found working, test it with actual data and check performance.

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding. The increasing visibility of software as a system element and attendant "cost" associated with a software failure is motivation forces for a well planned, through testing.

7.1 TESTING OBJECTIVES:

The testing objectives are summarized in the following three steps. Testing is the process of executing a program with the intent of finding an error. A good test case is one that has a high probability of finding an as yet undiscovered. A successful test is one that uncovers as-yet-undiscovered error.

7.1.1 TESTING PRINCIPLES:

All tests should be traceable to customer requirements. Tests should be planned long before testing begins, that is, the test planning can begin as soon as the requirement model is complete. Testing should be "in the small" and progress towards testing "in large". The focus of testing will shift progressively from progressively from programs, to individual modules and finally to the entire project. Exhaustive testing is not possible. To be more effective, testing should be one, which has highest probability of finding errors.

The following are attributes of good test:

- A good test has a high probability of finding an error
- A good test is not redundant
- A good test should be " best of breed"
- A good test should be neither too simple nor too complex

7.2 LEVELS OF TESTING:

The details of the software functionality tests are given below. The testing procedure that has been used is as follows

- Unit Testing
- Integration testing
- Validation testing
- Output testing

7.2.1 UNIT TESTING:

Unit testing is carried out to verify and uncover errors within the boundary of the smallest unit or a module. In this testing step, each module was found to be working satisfactory as per the expected output of the module. In the package development, each module is tested separately after it has been completed and checked with valid date. Unit testing exercises specific paths in the modules control structure to ensure complete coverage and maximum error detection.

7.2.2 INTEGRATION TESTING:

Integration testing address the issues associated with the dual problems of verification and program construction. After the software has been integrated a set of high-order test are conducted. The main objective in this testing process is to take unit tested modules and build a program structure that has been dictated by design.
The following are the types of Integrated Testing:

7.2.2.1 TOP-DOWN INTEGRATION:

This method is an incremental approach to the construction of program structure. Modules are integrated by moving downward through the control hierarchy, beginning with the main program module. The module subordinates to the main program module. The module subordinates to the main program module are incorporated to the structure in either a depth first or breath-first manner.

7.2.2.2 BOTTOM UP INTEGRATION:

This method designs the construction and testing with the modules at the lowest level in the program structure. Since the modules are integrated from the bottom up, processing required for modules subordinate to given level is always available and the need for stubs

is eliminated. The bottom up integration strategy may be implemented with the following steps:

- The low level modules are combined into clusters that perform a specific software sub-function.
- A driver (i.e.) the control program for testing is returned to coordinate test case input and output.
- The cluster is tested.
- Drivers are removed and clusters are combined moving upwards in the program structure.

7.2.3 VALIDATION TESTING:

At the end of integration testing, software is completely assembled as a package, interfacing errors have been uncovered and correction testing begins.

7.2.3.1 VALIDATION TEST CRITERIA:

Software testing and validation is achieved through a series of black box tests that demonstrate conformity with the requirements are achieved, documentation is correct and other requirements are met.

7.2.4 OUTPUT TESTING:

Output testing is a series of different test whose primary purpose is to fully exercise the computer based. Although each test has a different purpose all the work should verify that all system elements have been properly integrated and perform allocated functions.

Output testing is the stage of implementation, which is aimed at ensuring that the system works accurately and efficiently before live operations commence. The input screens, output documents were checked and required modifications made to suit the program specification. Then using the rest data prepared, the whole system was tested and found to be a successful one.

CONCLUSION AND FUTURE OUTLOOK

8. CONCLUSION AND FUTURE LOOK

8.1 CONCLUSION:

The complete design and development of the system is presented in this dissertation. The system has user-friendly features. It is possible for any user to use this system.

The programming techniques used in the design of the system provide a scope for further expansion and implementation of any changes, which may occur in the future. The system has been tested by connecting with many systems and they provide satisfactory performance.

This system is developed with the specifications and abiding by the existing rules and regulations of the company.

Since the requirements of any organizations and their standards are changing day by day the system has been designed in such a way that its scope and boundaries could be expanded in future with little modifications. As a further enhancement this system can be integrated with any other system.

This system has been developed using Visual C++. The main aim behind the development of this system is to provide a solution for the client to execute DOS commands through browser.

8.2 LIMITATIONS:

This project is very simple and easy to implement. This project is working properly. This project is a successful one and this can be implemented easily. The system developed can monitor only one system at a time. It can be run only in windows platform. The system can monitor only those systems connected in one server and not many servers. The future enhancement can monitor multiple systems at a particular time.

8.3 FUTURE ENHANCEMENTS:

There are many features that can be added to the system. Due to the insufficient time it could not be incorporated in this system.

The future enhancements that can be provided are:

- Access to multiple clients at a particular time.
- Monitoring the system of another server from a server.

REFERENCES

9. REFERENCES

1.David J. Kruglinski, George Shepherd, Scot Wingo,
"Programming Microsoft Visual C++",
Microsoft Press; Fifth Edition , 1992

2.Richard C. Leinecker and Archer Tom,
"Visual C++ 6 Programming Bible",
IDG Books India (P) Ltd; Fifth Edition. , 1999

3. Roger S. Pressman
"Software Engineering and Application",
McGraw Hill; Fourth Edition

4.John Paul Mueller
"Visual C++ 6 from the Ground up",
Tata McGraw Hill. , 1998

5.MSDN Library

Web-sites:

www.codesheaven.com
www.SDKnet.com
www.codeguru.com
www.microsoft.com

APPENDIX

10. APPENDIX

10.1 SAMPLE CODING

SERVER MODULE

```
//      Visual C++ Project | Settings | Link | Object/Library modules | ws2_32.lib

#define WIN32_LEAN_AND_MEAN
#include <winsock2.h>
#include <iostream.h>
#include <stdlib.h>

#include "student.cpp"
#include "pipe.cpp"
#include "network.cpp"

// STRnew - Returns a pointer to memory allocated of specified length
extern "C" char * STRnew(int length) {
        return (new char[length]);
}

// execute - Parse COMMAND and input from browser request. Execute and return
output
//      string. The broswer request is in one of two forms, always terminated by a space:
//      command=COMMAND<sp>
//      command=COMMAND+input1+input2+...<sp>
char * execute(char clientinput[]) {
        char *command, *s, *input="";
        int index1, index2;

        if((index1=STRindex("command=", clientinput)) != -1) {  //      Not      initial
connection
```

```
            index1=index1+8;
            index2=STRtoken(" \n\r",clientinput+index1);
            if (index2 != -1) {
                clientinput[index1+index2]='\0';                    //
Terminate GET input
                CHARrep(clientinput+index1, '+',' ');               // All + to ' '
                index2=STRtoken(" ",clientinput+index1);
                if(index2==-1)
                    index2=STRlen(clientinput+index1);              //  No
blank found
                else
                    index2++;
            // Remove blank
                input = STRnew(STRlen(clientinput+index1+index2)+1);
                STRcpy(clientinput+index1+index2, input);
                clientinput[index1+index2]='\0';                    //
Terminate after command

            }
            command = STRnew(STRlen(clientinput+index1)+1);
            STRcpy(clientinput+index1,command);
            STRtrim(command);
            s = EXECUTEcmd(command, input);
            delete(command);
            delete(input);
        }
        else {
            s=STRnew(1);
        // Must return valid string
            s[0]='\0';
        }
```

```
        return s;
}
// process - Process the input from the browser by:
//      1. Executing the user command request,
//      2. Wrapping the execution output between <pre> and </pre> for HTML formatting.
//      3. Adding the user text box for the next request.
char * process(char clientinput[]) {
        static const char *pre = "<pre>";
        static const char *_pre = "</pre>\n\n";
        static const char *title = "<H2><font color=\"#3366FF\">Remote Server</font></H2>Enter command.";
        static const char *action = "<form action=\"\" method=\"GET\">";
        static const char *input = "<input type=\"text\" name=\"command\" size=50>";
        static const char *form = "</form>\n\n";
        char *s, *t;

        s = execute(clientinput);
        t = STRnew(STRlen(s)+STRlen(pre)+STRlen(_pre)+STRlen(title)+
                        STRlen(action)+STRlen(input)+STRlen(form)+1);

        STRcpy(title, t);           // Add user input box
        STRcat(action, t);
        STRcat(input, t);
        STRcat(form, t);
        STRcat(pre, t);             // Pre-formatted output automatically

        STRcat(s,t);
        STRcat(_pre, t);
```

```
        delete(s);
        return t;
          }
// RemoteServer
void main(void)
{
        char                    *inbuffer, *outbuffer;
        SOCKET                  s, h;
   struct sockaddr_in  sin;
        int                     port=80;
cout << "Remote Server Running\n\n"; flush(cout);
if(startServer(port, s, sin) != 0) {
                cout << "startServer() failed with error " << WSAGetLastError() << '\n';
                return;
        }
        while(1) {
                if((h=connectClient(s,sin))==NULL) {     // Accept incoming
                connection
                        cout << "connectClient() failed with error " <<
                        WSAGetLastError() << '\n';
                        return;
                }
                inbuffer = receiveClient(h);
                cout << "\nServer Input\n" << inbuffer << "\n"; flush(cout);
                outbuffer = process(inbuffer);
                cout << "Server Output\n" << outbuffer << "\n\nCtrl C to stop server\n";
                flush(cout);
                sendClient(h,outbuffer);
                delete(outbuffer);
                delete(inbuffer);
        }
```

PIPE MODULE

```
#include <windows.h>

#define BUFSIZE 4096

HANDLE hChildStdinRd, hChildStdinWr, hChildStdinWrDup,
  hChildStdoutRd, hChildStdoutWr, hChildStdoutRdDup,
  hInputFile, hSaveStdin, hSaveStdout;

BOOL CreateChildProcess(char cmd[]);
char *WriteToPipe(char input[]);
char *ReadFromPipe(VOID);
char *errorMessage(char s[]);

char * DOS(char cmd[], char input[])
{

  SECURITY_ATTRIBUTES saAttr;
  BOOL fSuccess;
  char *inputCopy = new char[strlen(input)+3];
  char *writeResult;

  strcpy(inputCopy, input);
  strcat(inputCopy, "\r\n");    // Add \n\r to terminate any input

// Set the bInheritHandle flag so pipe handles are inherited.

  saAttr.nLength = sizeof(SECURITY_ATTRIBUTES);
  saAttr.bInheritHandle = TRUE;
  saAttr.lpSecurityDescriptor = NULL;
```

```
  // The steps for redirecting child process's STDOUT:
  //     1. Save current STDOUT, to be restored later.
  //     2. Create anonymous pipe to be STDOUT for child process.
  //     3. Set STDOUT of the parent process to be write handle to
  //        the pipe, so it is inherited by the child process.
  //     4. Create a noninheritable duplicate of the read handle and
  //        close the inheritable read handle.

// Save the handle to the current STDOUT.

  hSaveStdout = GetStdHandle(STD_OUTPUT_HANDLE);

// Create a pipe for the child process's STDOUT.

  if (! CreatePipe(&hChildStdoutRd, &hChildStdoutWr, &saAttr, 0))
    return errorMessage("Stdout pipe creation failed\n");

// Set a write handle to the pipe to be STDOUT.

  if (! SetStdHandle(STD_OUTPUT_HANDLE, hChildStdoutWr))
    return errorMessage("Redirecting STDOUT failed");

// Create noninheritable read handle and close the inheritable read
// handle.

   fSuccess = DuplicateHandle(GetCurrentProcess(), hChildStdoutRd,
     GetCurrentProcess(), &hChildStdoutRdDup , 0,
     FALSE,
     DUPLICATE_SAME_ACCESS);
  if( !fSuccess )
     return errorMessage("DuplicateHandle failed");
```

```
    CloseHandle(hChildStdoutRd);

    // The steps for redirecting child process's STDIN:
    //     1. Save current STDIN, to be restored later.
    //     2. Create anonymous pipe to be STDIN for child process.
    //     3. Set STDIN of the parent to be the read handle to the
    //        pipe, so it is inherited by the child process.
    //     4. Create a noninheritable duplicate of the write handle,
    //        and close the inheritable write handle.

  // Save the handle to the current STDIN.

    hSaveStdin = GetStdHandle(STD_INPUT_HANDLE);

  // Create a pipe for the child process's STDIN.

    if (! CreatePipe(&hChildStdinRd, &hChildStdinWr, &saAttr, 0))
      return errorMessage("Stdin pipe creation failed\n");

  // Set a read handle to the pipe to be STDIN.

    if (! SetStdHandle(STD_INPUT_HANDLE, hChildStdinRd))
      return errorMessage("Redirecting Stdin failed");

  // Duplicate the write handle to the pipe so it is not inherited.

    fSuccess = DuplicateHandle(GetCurrentProcess(), hChildStdinWr,
      GetCurrentProcess(), &hChildStdinWrDup, 0,
      FALSE,                  // not inherited
      DUPLICATE_SAME_ACCESS);
    if (! fSuccess)
```

```
    return errorMessage("DuplicateHandle failed");

  CloseHandle(hChildStdinWr);

// Now create the child process.

  if (! CreateChildProcess(cmd)) {    // Program execution failed, try as DOS command
        if (! CreateChildProcess("cmd.exe"))        // NT cmd.exe failed, try
command.com
              if (! CreateChildProcess("command.com"))  // Win98/98 failed also
                    return errorMessage("Cmd.exe and Command.com execution
failed\n");
              char *e="\n\rEXIT\n\r";
              char *s = new char[strlen(inputCopy)+strlen(cmd)+strlen(e)+2];
              strcpy(s,cmd);
              strcat(s," ");
              strcat(s,inputCopy);
              strcat(s,e);
              delete(inputCopy);
              inputCopy=s;
  }

// After process creation, restore the saved STDIN and STDOUT.

  if (! SetStdHandle(STD_INPUT_HANDLE, hSaveStdin))
    return errorMessage("Re-redirecting Stdin failed\n");

  if (! SetStdHandle(STD_OUTPUT_HANDLE, hSaveStdout))
    return errorMessage("Re-redirecting Stdout failed\n");

// Write to pipe that is the standard input for a child process.
```

```
  if ((writeResult=WriteToPipe(inputCopy)) != NULL)
        return writeResult;
  delete(inputCopy);

// Read from pipe that is the standard output for child process.

      return ReadFromPipe();

}


BOOL CreateChildProcess(char cmd[])
{
  PROCESS_INFORMATION piProcInfo;
  STARTUPINFO siStartInfo;

// Set up members of STARTUPINFO structure.

  ZeroMemory( &siStartInfo, sizeof(STARTUPINFO) );
  siStartInfo.cb = sizeof(STARTUPINFO);

// Create the child process.

  return CreateProcess(NULL,
    cmd,                        // command line
    NULL,       // process security attributes
    NULL,       // primary thread security attributes
    TRUE,       // handles are inherited
    0,          // creation flags
    NULL,       // use parent's environment
```

```
    NULL,        // use parent's current directory
    &siStartInfo,  // STARTUPINFO pointer
    &piProcInfo);  // receives PROCESS_INFORMATION
}

char * WriteToPipe(char input[])
{
  DWORD dwWritten;

  WriteFile(hChildStdinWrDup, input, strlen(input), &dwWritten, NULL);

// Close the pipe handle so the child process stops reading.

  if (! CloseHandle(hChildStdinWrDup))
    return errorMessage("Close pipe failed\n");
  else
          return (NULL);
}

char *ReadFromPipe(VOID)
{
  DWORD dwRead;
  CHAR chBuf[BUFSIZE];
  HANDLE hStdout = GetStdHandle(STD_OUTPUT_HANDLE);
  char *newReturnBuffer, *returnBuffer=new char[1];
  returnBuffer[0]='\0';

// Close the write end of the pipe before reading from the
// read end of the pipe.

  if (!CloseHandle(hChildStdoutWr))
```

ASSEMBLY LEVEL LANGUAGE CODING FOR USER DEFINED COMMANDS

LOWCASE CONVERSION

```
CODE    Segment

   Main   Proc       Far
          Assume     Cs:CODE

     do:   Mov       Ah, 7
           Int       21H
           Or        Al, 20h
           Mov       Dl, Al
           Mov       Ah, 2
           Int       21H
  while:   Cmp       Dl, ' '
           Jne       do

           Mov       Ah, 4ch
           Int       21H
   Main    Endp

   CODE    Ends

           End    Main
```

VERSION COMMAND

```
.386
.Model  Compact, C

Extrn STRnew:near, STRcpy:near, STRlen:near
Public  VERSION

.DATA
versionstr      db      "<h2>Remote Server Version 2000</h2>",0
.CODE
VERSION  Proc    Near           ; extern "C" char * VERSION(void){
      Push      eBp             ;    static char *versionstr = "<h2>Remote Server Version
2000</h2>";
      Mov     eBp, eSp        ;  char *retstr = STRnew(STRlen(versionstr)+1);
                        ;  STRcpy(versionstr, retstr);
                        ;  return retstr;
                        ; }
      Push    Offset versionstr
      Call    STRlen
      Add     eSp, 4
     Add     eAx, 1
      Push    eAx
      Call    STRnew
      Add     eSp, 4
                        ; eAx points to memory
      Push    eAx             ; Save eAx before STRcpy
      Push    eAx
      Push    offset versionstr
      Call    STRcpy
      Add     eSp, 8
```

```
    Pop    eAx            ; Restore eAx after call to STRcpy

    Mov    eSp, eBp
    Pop    eBp
    Ret
VERSION  Endp
    End
```

FEW MICROSOFT DOS COMMANDS

ARP-The arp command is used to change the IP address on a network card

DOS-Date can be used to look at the current date the machine is set to as well as changed if new date is entered.

CLS-Cls is a command which allows the user to clear the complete contents of the screen and leave only a prompt.

ECHO-Echo is used to repeat the text typed in back to the screen and or can be used to echo to a peripheral on the computer such as a COM port

DIR-The dir command allows you to see the available files in the current and or parent directories.

Keyb-Keyb is used to change the layout of the keyboard used for different countries.

LABEL-Label is used to label the computer

MEM-Allows you to determine the available, used and free memory.

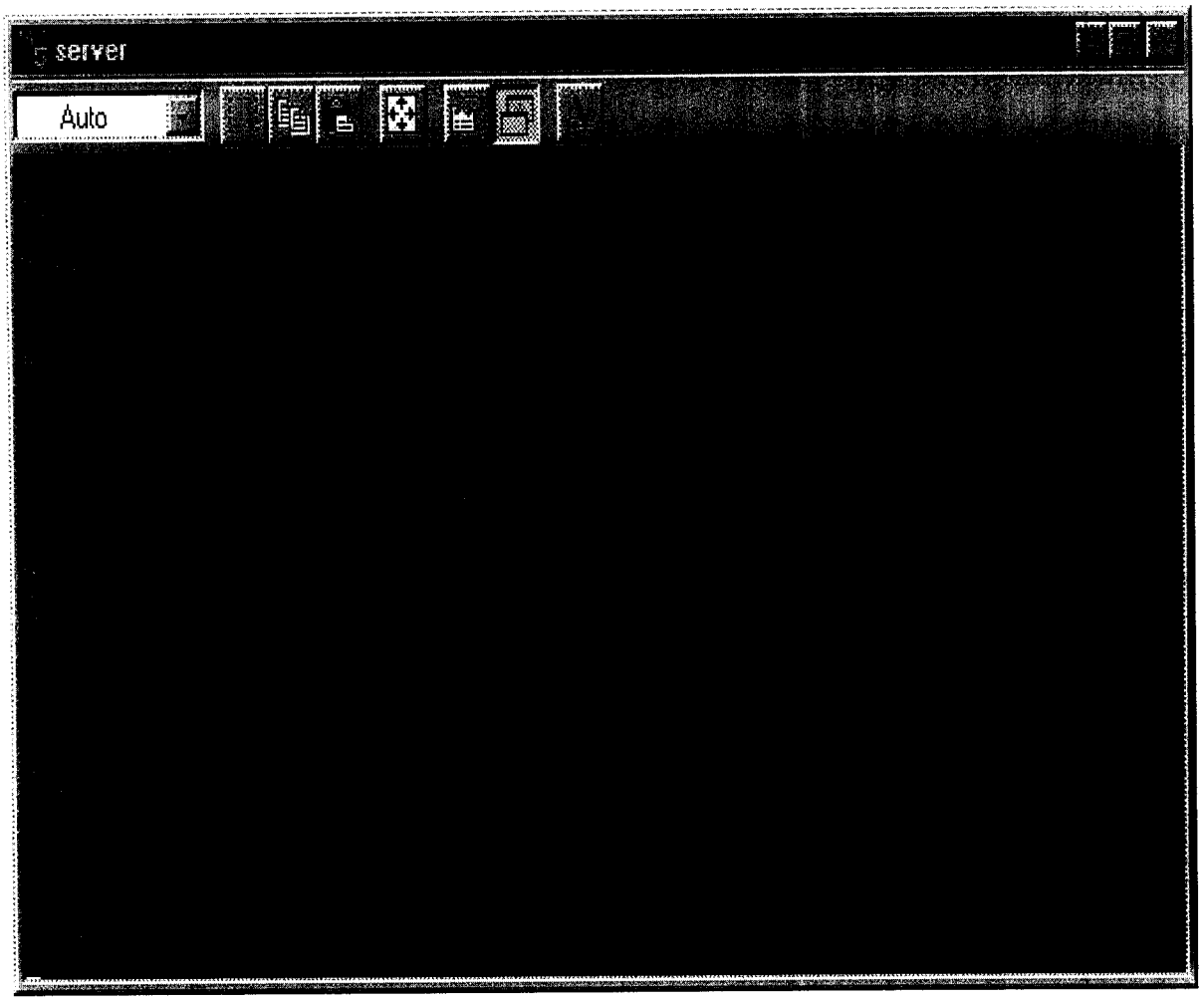
RENAME-Used to rename files and directories from the original name to a new name.

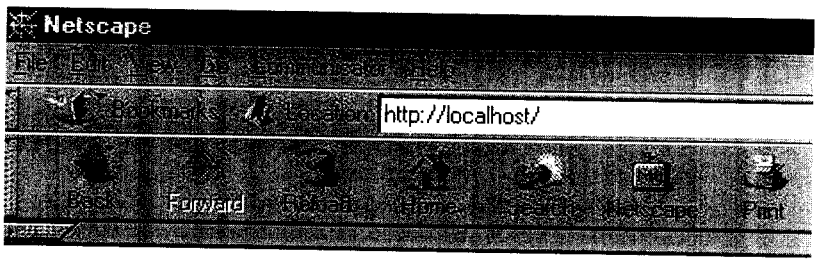
TREE-Allows the user to view a listing of files and folders in an easy to read list

Remote Server

Enter command.

```
Microsoft(R) Windows 95
   (C)Copyright Microsoft Corp 1981-1996.

C:\d\C335\C\server>DIR  *.cpp


 Volume in drive C has no label
 Volume Serial Number is 145C-12EB
 Directory of C:\d\C335\C\server

SERVER   CPP         6,626  11-10-99 10:19p server.cpp
PIPE     CPP         6,037  11-10-99  8:30p pipe.cpp
NETWORK  CPP         2,080  11-10-99  7:45p network.cpp
         3 file(s)         14,743 bytes
         0 dir(s)       2,059.37 MB free

C:\d\C335\C\server>EXIT
```

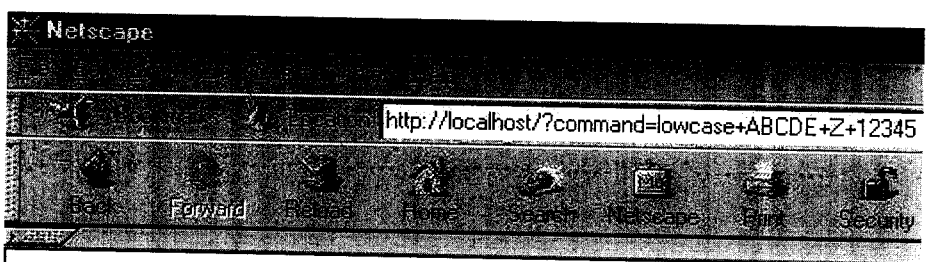

server
Auto



Remote Server

Enter command.

```
lowcase ABCDE Z 12345
```



Remote Server

Enter command.

abcde