1753

Remote Login With File Transfer Facility

Dissertation submitted to
JAWAHARLAL NEHRU UNIVERSITY
in partial fulfilment of requirements
for the award of the degree of
Master of Technology
in
Computer Science

by

MUKESH KUMAR

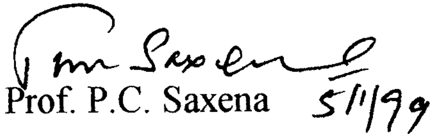
42 p+ Source code

School of Computer & Systems Sciences
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI - 110 067

CERTIFICATE

This is to certify that the dissertation entitled ***"REMOTE LOGIN WITH FILE TRANSFER FACILITY"*** which is being submitted by **Mr.MUKESH KUMAR** to the School of Computer & System Sciences, Jawaharlal Nehru University, for the award of **Master of Technology in Computer Science**, is a record of bonafide work carried out by him.

This work is original and has not been submitted in part or full to any university or institution for the award of any degree.



Prof. P.C. Saxena 5/1/99

(Dean SC&SS)

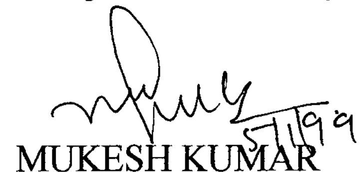
Dr.R.C.Phoha

(Supervisor)

DECLARATION

This is to certify that the dissertation entitled ***"REMOTE LOGIN WITH FILE TRANSFER FACILITY"*** which is being submitted to the School of Computer & System Sciences, Jawaharlal Nehru University, for the award of **Master of Technology in Computer Science**, is a record of bonafide work carried out by me.

This work is original and has not been submitted in part or full to any university or institution for the award of any degree.



MUKESH KUMAR

ACKNOWLEDGEMENT

I wish to convey my sincere gratitude and acknowledgement to my guide Dr. R.C. Phoha, School of Computer & Systems Sciences, Jawaharlal Nehru University for his wholehearted, tireless and relentless efforts in helping me throughout in completing this project.

I would like to record my thanks to my Dean, Prof. P. C. Saxena, School of Computer & Systems Sciences, Jawaharlal Nehru University for providing the necessary facilities in the centre for the successful completion of this project.

I take this opportunity to thank all of my faculty members and my brother Rakesh kumar & Priyanka friends for their help and suggestions during the course of my project work.

MUKESH KUMAR

dedicated to my beloved Parents...

CONTENTS

ABSTRACT 4

1. INTRODUCTION 5

1.1 Networks 6

1.2 Client - Sever Model 9

1.3 Remote Logic 10

1.4 File Transfer Facility 11

2. MOTIVATION 12

3. REMOTE LOGIC 15

3.1 Introduction 16

3.2 Terminal Line Disciplines 17

3.3 Pseudo -Terminals 18

3.4 Terminals- Modes 19

3.5 Control Terminals 20

3.6 Remote Login Overview 21

3.7 Windowing Environments 21

3.8 Flow Control 25

3.9 Pseudo - Terminal Packet Mode 25

4. FILE TRANSFER 29

4.1 Introduction 30

4.2 File Transfer Packets 31

4.3 Security 32

4.4 Data Formats 33

4.5 Client User Interface 33

CONCLUSIONS 34

BIBLIOGRAPHY 36

APPENDIX 37

A. USERS MANUAL & GUIDE 38

A.1 Getting Started 38

A.2 How to Log Out ? 38

A.3 About ~ Command 38

A.4 About File Transfer 39

B. ABOUT FILES IN SOURCE CODE 40

REMOTE LOGIN WITH FILE TRANSFER FACILITY

ABSTRACT

This project "slogan : Remote Logic with File Transfer facility" provides the user of UNIX , the ability to logic onto a remote computer and also the facility of transferring the files between the remote and local computers. Through the services remote logic and file transfer protocol are already provided, the project attempts to merge the features of both facilities remote logic and file transfer protocol ignored to eliminate the difficulties encountered while using them.

Highlights of the project :

- **.Remote logic facility :--**
  The ability to logic onto a remote computer and work on it.

- **File Transfer facility:--**
  The ability to transfer files(i.e. getting and sending files) between the local and remote machines. Transfer of files can be done in two modes ASCII and binary modes.

- **Local flow control:--**

- **Local commands execution :--**
  The facility to execute any local command is also provided.

Introduction

INTRODUCTION

1.1 Networks

Computer network have revolutionized our use of computers. They pervade our every life, from automated teller machines, to airline reservation system, used to be stand --alone entities. each computer was self contained and had all the peripherals and software required to do a particular job. If a particular feature was needed such as line printer output, a line was attachedtothe computer If large amounts of disk storage were needed, disks were added to the system. what helped change this is the realization that computers and there users need to share information and resources.

Information sharing can be electronic mail or file transfer. resource sharing can be involve accessing a peripheral on another system. Twenty years age this type of sharing took place by exchanging magnetic tapes, decks of punched cards and line printer listings. Today computers can be connected together using various electronic techniques called network network can be as simple as two personal computer connected together using a 1200 baud modem, or as complex as the TAP/PI interment, which connects over 150,000 systems together. the number of ways to connect a computer to network Armenia, as are the various things we can do once connected to a network. some typical network applications are:

- Exchange electronic mail with users on the computers. It is commonplace these days for people to communicate regularly using electronic mail.

- Exchange files between systems. for many application it is just as easy to distribute the application electronically, instead of mailing diskettes or magnetic tapes. file transfer across a network also provides faster delivery.

- Share peripheral device. Examples range from sharing of the line printers to the sharing of magnetic tape derives. A large push towards the sharing of peripheral device has come from the personal computer and work station market, since often the cost of a peripheral can exceed the cost of the computer in an organization with many personal computer or workstations sharing peripherals makes sense.

- Execute a program on another computer. There are cases where some other computer is better suited to run a particular program. For example, a time sharing system or a work station with good program development tools might be the best system on which to edit and debug a program . Another system, however. might be better

equipped to run the program . this is often the case with the programs that required the special features, such as parallel processing or vast amount of storage
The National science Foundation(NFL) has connected the six super computer centers using network, allowing scientists to access these computers electronically. Years ago, access to facilitates such as this would have required mailing decks of punched cards or tapes.

- Remote login. If two computers are connected using a network, we should be able to one from the other (assuming we have an account on both system) It is usually easier to connect computers that are using a net work network and provide a remote login application. than to connect every terminal in an organization to every computer.

1.1.1 Layering

Given a particular task that we want , such as providing a way to exchange files between two computers that are connected with a network. we divide the task into pieces and solve each piece. In the end we connect the pieces back together to form a final solution. we could write a single monolithic system to solve the problem in pieces leads to and more extensible solution.

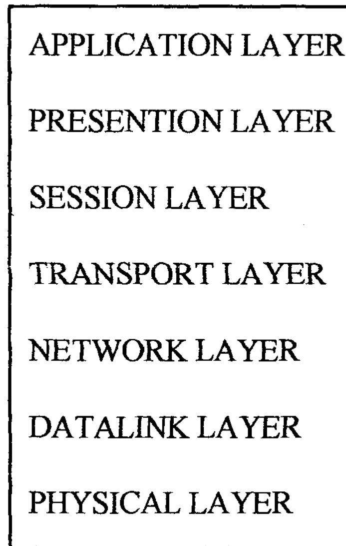
It is possible that part of solution for a file transfer program can also be used for a remote printing program is unable for computer connected with a leased telephone line. In the context of networking ,this is called layering. we divide the communication problem into pieces (layers) and let each (layer) and let each layer concentrate on providing a particular function. well--defined interfaces are provided between the layers.

The principle to be applied :

1. A layer should be created where a different level of abstraction is needed
2. Each layer should perform a well defined function.
3. The function of each layer should be chosen with an eye toward defining internationally standardized protocols.
4. The number of layers should be large enough that distinct functions need not be thrown together in the same layer out of necessity, and small enough that the architecture dose not become unwisely.

1.1.2 THE OSI MODEL

The international standards orgnazation (OSI) decided upon a protocol known as open System Interconnection (OSI). this concept was intend to be of relevance in realm of data communication , bring about a standard in the world of networking.
The OSI model comprises of 7 layers:



The Physical layer at the lowest level, concerns itself with the bare hardware. It is concerned transmitting raw bits over a communication channel.

The Data link layer ensures error--free data delivery . it is contingent on the physical medium. No universal protocol exists at this level.

The network layer is concerned with controlling the operation of the subnet. A key design issue is determining how packets are routed from source to destination.

The Transport layer monitors the service of the 3 foregoing layers vis-à-vis transmission media and information flow. it functions like a pipe between the user and the above 3 layers .

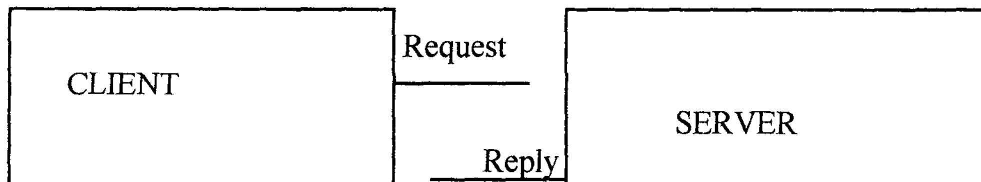
The Session layer controls communication between the application and synchronizes data exchange .

The presentation layer formats the data in a way comprehensible to the application and to the application.

The Application layer is at highest running of the ladder. Communication here is handled at the semantic level. The issue of inter-process communication, file transfer and broadcast communication fall under its aegis.

1.2 Client-- Server Model

The standard model for net work application is the client--server model a server is a process that is waiting to be contacted by aclient process so that the server can do some thing for the client.



A typical (but not mandatory) scenario is as follows :

The server process is started on some computer system . It initializes itself, then goes to sleep waiting for a client process to contact it requesting some service.

A client process is started, is either on the same system or another system that is connect to the server's system with a network. Client process are often initiated by an interactive user entering a command to atime shareing system. The client process sends a request across the network to the serve requesting service of some form.

Some examples of the same type of service that a server can provide are:
1. return the time of day to the client,
2. print afile on a printer
3. read or write a file on the server's system for the client,
4. allow the client to login to the server's system,
5. execute a command for the client on the server's system.

When the server process has finished providing its service to the client, the server gose to sleep, wating for the next client request to arrive.

1.3 Remote login

Remote login is an application which provides the facility of logging onto a remote computer and working on it. It provides the user , the feeling as if he is actually working on the remote computer. Atypical remote login session would be :

1.The user invokes the remote login service by typing at the command line prompt
2. The remote login client program start executing on the local computer as a result of the user invoking it
3.The remote login client process tries to get connected to the remote login server process running on the remote computer .
4.If connection is refused or is not established ,then the client process running on the local computer terminates by giving an error message stating connection not established
5.If connection is established between the client process on the computer and the server processes on the remote computer then the server asks for the login and password
6.The server verifies the validity of the login and password sent
7.If login and password are incorrect the connection is closed after sending an error message
8Then the user shell on the remote computer is executed by the login program
9The user on the local computer can start working on the remote m/c

1.4 File Transfer Portico

File Transfer protocol (FTP) is an application which provides the facility of transferring files (i.e. getting and sending files) between the local and remote computer. It also provides security by validating the user by his login and password remote and computer and allowing access to the files only which can accessed by the remote user. atypical ftp session would be :

1Theuserinvoke the FTP service by typing
the command line prompt
2TheFTPclient program starts executing
on the local computer as a result of the user
invoking it.
3 The ftp client processes tries to get connected
to the FTP server process running on remote
computer
4 If connection is refused or is not established
then the client process running on the local
computer by giving an error message starting
connection not established
5 If connections is established between the client
processes on the local computer and server process
on the on the remote computer then the server
ask for the login and password
6 The server verifies the validity of the login and
password sent
7.If login and password are incorrect the connection
is closed after sending an error message
8The user change to directory in which the wish
to get or send files
9 The user sends (gets) a file to (form) the remote
computer
10The user exit from the FTP session

Motivation

2. MOTIVATION

Remote Login is an application which provides the facility of logging onto a remote computer and working on it. It provides the user, the feeling as if he is actually working on the remote computer. Remote login service also provides local flow control.

File Transfer protocol is an application which provides the facility of transferring files between the local and remote computers. It also provide security by allowing access to files only which can be accessed by the remote user.

Let us consider following situation :

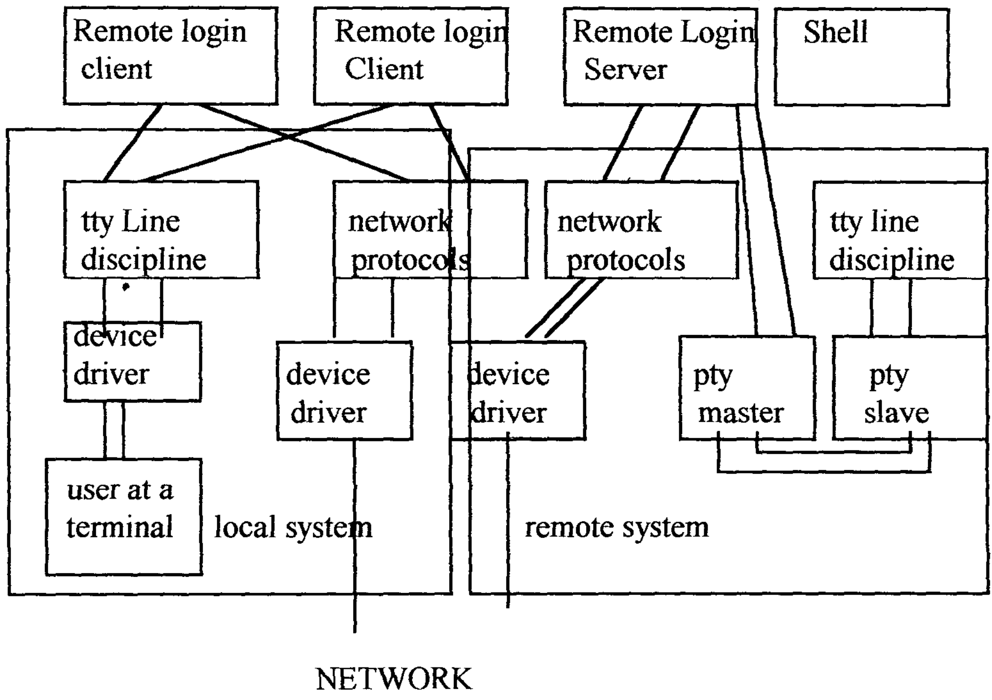
A user wishes to get a file from the remote machine. But he dose not know the actual location of the file (i.e the directory containing the file) or he dose not know the actual location of the file after viewing it. or He knows the file name but he does not know, which file out of many files with the same file name on the remote m/c, he wants.

In this situation the user follows the method shown on the next page:

1 Theuser executes the remote login application on the local computer
2Connection is established between the local and remote computer through the client and server processes running on the local and remote computers respectably
3The user search the dictory to find the file he wants
4The user confirms the filename by the file contents

1Theuser execute FTP application on the local computer
2Connection is established between the local and remote computers through the client and the server processing running on the local and remote computers respectively.
3 The user changes to present Working directory to the Directory in which the file is present
4 The user transfer the file and string

Generally logging on to a remote m/c (i.e. establishing a connection between local



Remote login client
Remote login Client
Remote Login Server
Shell
tty Line discipline
network protocols
network protocols
tty line discipline
device driver
device driver
device driver
pty master
pty slave
user at a terminal
local system
remote system
NETWORK

and remote m/cs) takes time and this delays our main purpose of transferring the file. Thus the method discussed above , in getting the file, is the consuming and we have to have a better method of solving this problem.

The problem discussed above provided the motivation for this project. this project is designed with eliminating the difficulties encountered and provides a way in which the user can login onto the remote computer and simultaneously.

- Work a: the remote place.
- Transfer files between local and remote computers.

Remote Login

3. Remote login

3.1 Introduction

The ability to login from one computer to another is an important net work application. there are two remote login applications: rlogin, which assumes the server is another

Unix system, and telnet, Which is a standard internet application that most TCP/IP support the remote login protocol.

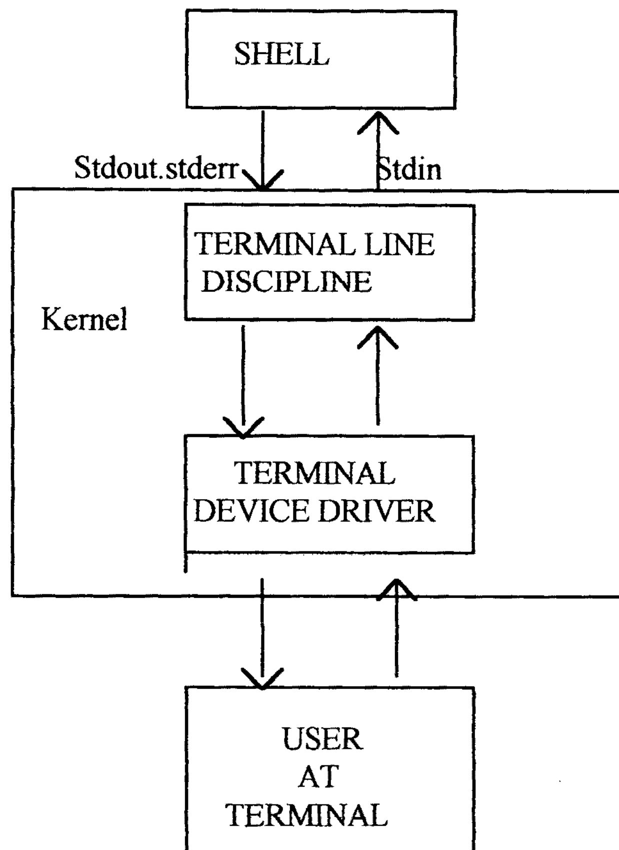
The purpose of the REMOTE LOGIN protocol is to provide is a bi directional, eight - bit byte oriented communication facility. Its primary goal is to flow a standard methods of interfacing terminals devices and terminal-- oriented process to each other.

It is envisioned that the protocol may also be used for terminal--terminal communication (" linking ") and process--process communication (distribution computation).

Any discussion of the remote login from one computer system to another involves the details of the terminal handling. The discussion also requires knowledge and pseudo terminals. Pseudo terminals have always been a mystical and undocumented feature of unite. but to understand the use of a network to login to another computer, knowledge of pseudo -terminals is required.

3.2 Terminal Line Discipline

Terminal devices are complicated by the line discipline is with inthe kernel ,some where between the actual device driver and the user process. the following figure shows this for a normal interactive shell.



There are several functions that can be done by a line discipline module

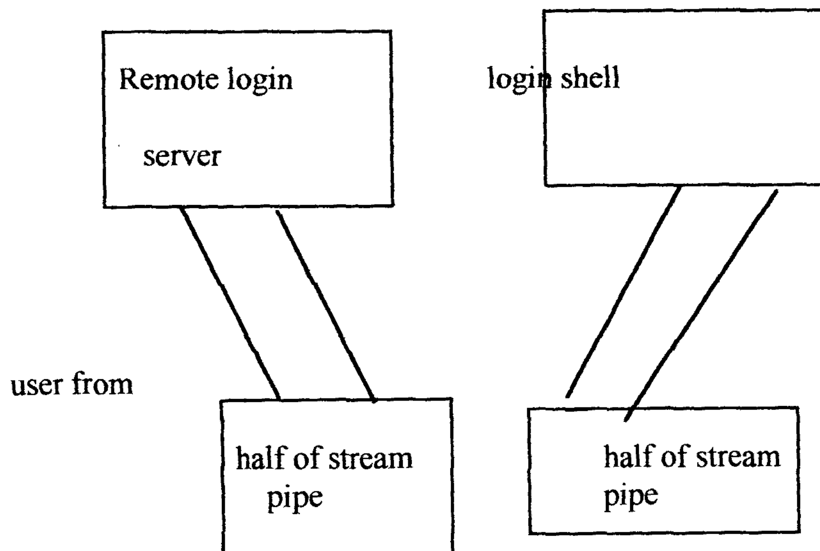
1. Echo the characters that are entered .
2. Assemble the characters entered into lines, so that a process reading from the
3. terminal receives complete lines.
4. Edit the lines that are input. Unix allows you to erase the preceding character and also to kill the entire line being input and start over with anew line.
5. Generate singnals when certain terminals keys are entered. The SIGINT and SIGQUIT signals can be generated this way, for example.
6. Process flow controls characters. For example press the Control-S key, the out put to the terminal is stopped. To restart the out put, the Control -Q key is entered.
7. 6.Allow you to enter an end of-file-character.
8. Do character conversions . for example ,every time a process writes a new line character, the line discipline can convert it is to carriage return and a line feed . Also, tab characters that are output can be converted to spaces if the terminal doesn't handle tab characters.

Part of the complication of the terminal handling arises from the many different devices that can be connected to an asynchronous serial line on a computer. not only are interactive terminals connected this way, but printer's modems, plotters, and the like are also attached to the terminals line is being used for interactive input, different programs want to access the terminal difficulty . some process lines some are full screen editors, some want the echo facility disabled (when entering a password, for example).

3.3 Pseudo--Terminals

Our remote login server process forks a child process and executes the login in the child process. The server process (running on the remote computer) should take the input of the user from the client process (running on the computer) and should give it as input to the login shell process. And the server process should take the output of the login shell process and should send it to the client process which outputs on the local computer) A pipe could help us in the inter process communication between the server process and login shell process .

A view of the above description :

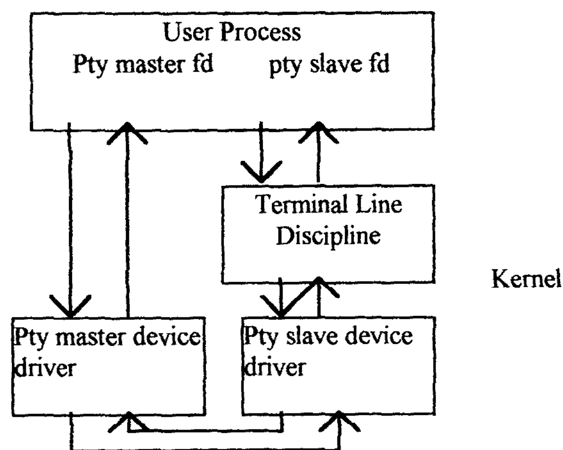


There are, however, some problems. When any of the shell start up, they check their standard input and standard out put to see if both belong to a terminal device, using

the standard Unix function is atty. Since both of these descriptors for the shell refer to the stream pipe that is created between the shell and the recording process , the isatty function returns false for both descriptors. we do not get the prompt We the prompt .Also. we can not run the program that we requires a terminal for example, if we run the vi editor, the edited generates an error message saying that it requires an addressable cursor. Finally if we execute the Unix command tty, which prints the path name of the terminal device being used, it prints the message "dot a tty"

The problem is that the communication channel between the server process and the login shell, dose not look like a terminal device to the system . If there is some way to put a block containing the " terminal line discipline" between the server process and the login shell, these problems disappear. Indeed, this is what pseudo--terminals are designed to do. Doing this the shell thinks that it is talking to a terminal. And vi editor works, since it can execute the terminal ioctls that it needs for full screen control. And for full screen control. And all the problems, we have got, when we used a stream pipe are gone.

A Pseudo--terminal is a pair of devices. One half is called the master and the other half is called the slave process open a pair of pseudo --terminal devices and gets two file descriptors. The slave portion of a pseudo--terminal present an interface to the user process that looks like a terminal device. We can picture this as shown in the fig.



We use the term "pty" as an abbreviation for pseudo --terminal Anything written to the master pty is looped around and appears as input from the slave pty. Similarly any thing written to the slave pty appears as input from the master pty.

3.4 Terminals Modes

There will be two terminal line disciplines between the actual terminal and login shell process, one at thc local computer and another will be between the pseudo slave device and the login shell on the remote computer . we only want a single line discipline module interpreting the character entered on a terminal we want the line discipline module associated with the slave 's pseudo -terminal to do the normal terminal processing . So we need to initialize the mode of the pseudo terminal to be identical to the mode of the actual terminal . and we need to put terminal line discipline module in raw that just passes every character through to the server process.

A terminal device can be one of the three mobes

- **Cooked mode** provides all the processing step listed in the terminal discipline section . the input is collected into lines and all special character processing is done (erase processing , signal generation , etc) This is the normal mode for interactive use.

- **Row mode** lets the process receive every character as it is input , with no interpretation done by the system . Raw mode is used, for example , by full screen editors such as vi, and also by programs that use a serial line for some thing other than interactive use . An example of the latter is UUCP . A classic problem with raw mode is if a process enables this mode but terminates without changing the mode back to cooked mode. You are typically left with a "raw" terminal –echoing is probably disabled and you might have to enter the line feed key. Instead of the normal return key to end a line , for example.

TH-7651

- **Check mode** is some where between cooked mode. The cbreak mode provides character at a time to the process reading from the terminal, instead of collecting the input into the lines . The signal --generating keys are still processed, the editing features are disabled.

3.5 Control Terminals

Control terminal is one which is used for dispatching signals, when certain terminal keys are pressed and when alogin shell terminates. The terminal group ID identifies the control terminal for a process group . Under system V, a process disassociates from itself from its control terminals (if it has one) by calling the stepgrp system call. An example which requires this disassociation , is adaemon process
In our application , we want the shell associates with the pseudo terminal slave as its control terminal. At the same time , we want that shell process to disassociate from the terminal.

3.6 Remote Login Overview

Let's first show a picture of all the process involved in the remote login client and server. This is shown in the figure on the next page. We will refer to the system that you initially login to as either the local system or client system, The system that remotely login to is the remote system or server system .

The terminal line discipline on the local system is placed into the raw mode with echoing disabled on the local system is placed into the raw process, so that all keystrokes are passed to the remote system. As we saw earlier, The raw mode is required to run programs such as Vi editor on the remote system. In the normal Unix fashion, character that are entered on the local system are echoed by the remote system . If the remote system is in a raw mode (such the vi editor is being run on the remote system the remote system), then the echoing is done by that remote process itself (e.g vi) regardless of which box on the remote system is doing the echoing, every character that is echoed on your terminal has to go from the client system. through the network to the remote system , and then back , before being echoed on your terminal

Notice that the remote login client process forks so that two processes are running on the local system, each process handling the flow of data in a single direction of data flow . the select system call is used by the server to multiplex its two streams. Also there has to be some from of information flow between the parent and child client process without stopping the child process . By doing this, we can stop the parent . allowing you to enter other commands on the client system , while still allowing any out put from the remote system to appear on your terminal.

The 4.3BSD manual page describes the remote login facility as " remote -echoed , locally flow -controlled, with proper back flushing of output".

3.7 Windowing Environments

The typical method used by Unix to specify terminal' features is to have a data file of terminal characteristics. 4.3bsd and early versions of system V use the termcap file for this, while more resent versions of system V use the terminof file. Contained in either of these files for each terminal type the size of terminal 's screen . Typical values are 24 lines by 80 columns a problem with these terminal capability file that they assume that the size of the terminal window doesn't change the size of a terminal window doesn't change . current technology however provides a variety of ways for you to change the size of a terminal's window dynamically during alogin session . software that uses the full capabilities of the screen, such as a screen, such as a full screen editor, has to be made aware of any changes in a

windows's size so they can redraw the screen.

The ability to have a terminal device support multiple windows of varying sizes and to allow you to change the size of a window during a login session is not restricted to bit mapped displays . 4.3 BSD, for example , provides the window program that provides these capabilities on standard ASCII terminals.

To support a windowed environment , the current size of a window has to be stored in a central location during a login session . Also , there has to be some way to allow process to read the current size , the current size and be made aware whenever the size changes 4.3Bsd provides ioctls and store the window size.

```
#include <ioctl.h>

        int ioctl (int fd, tiocgwinsz, struct winsize * winptr0;/* get*/

        int ioctl (int fd, Tiocswinsz, struct winsize* winptr);/* set */
struct winsize {

unsigned short ws_row      /* rows, in characters */
unsigned short ws_ col     /* columns, in characters */
unsigned short ws_ xpixel; /* horizontal size, pixels */
unsigned short ws_ypixel;  /* vertical size, pixels */
};
```

The kernel maintains a witness a winsize structure for each terminal and pseudo - terminal, but dose not it for anything . All the kernel dose is provide a central location for active processor to keep track of a window's size.

The kernel also generates the SIGWINCH signal whenever the size of a window changes. this signal is sent to the terminal process group associated with the terminal as an example , the 4.3BSD vi editor cathes the SIGWINCH signal This editor needs to know the size of the terminal's window so that it can wrap long lines from one line to the next , and to know where the bottom line of the screen is (its command and output line) Whenever the size of the window changes, It catches the signal and redraw the screen .

When a windowing environment is being used , such as the 4.3 BSD window command that we mentioned above , pseudo--terminals are typically used to provide one login shell for each active window . under 4.3BSD, we have created two windows. we have arrangement as shown in the below

The layers process is the host process that handles the multiplexed terminal. It communicates over a single RS-232 line with the terminal. It also communates with the processes running in each window (layer) using a pseudo terminal . When you resize a window , the operation is done by the firm ware in the terminal. (a small operating system that executes in the terminal) this firmware sends a special sequences of bytes across the RS-232 line to the layers process . The layers process figures out which window was resized from the sequence of bytes it receives and execute the TIOCSWINSZ ioctl to set the new window size. This system call is applied to the master pseudo -terminals associated with the window that resized . The Kernel then sends the SIGWINCH signal to the corresponding slave side of the pseudo terminal.

Not shown in the figure is the login shell that was used to invoke the layer process. This login shell is waiting for all the layers process to terminate. Now let's consider what effect resizing a window has on a remote login session . The problem is that resizing a window on the client 's system . Consider the layers system . shown in the figure and assume we have an remote login client running in one window . This gives us the processes shown in the figure below. The five steps that have that have to be taken by the remote login client and server are shown in the figure shown.

1. You resize the window which causes a special sequence of bytes to be sent to the layers process

2. The layer processes recognizes the sequence of character as a window resize command and issue an ioclt of TIOCSWINSZ for the pseudo terminal master device This causes the SIGWINCH signal to be sent to the process group of the pseudo terminal slave device , the remote login process

3. The remote login parent process catches the SIGWINSZ signal .It then issues an ioctl of of TIOCGWINSZ to fetch the new window size the new size of the window of the window is then sent across the network connection to the remote login server process .

4. The remote login server receives the new window size from the client and issues an ioctl of TIOCSWINS7 on its pseudo-terminal master device. The SIGWINCH signal is then sent by the kernel to the kernal to the process group of the pseudo-terminal slave device,the vi editor in the example.

As shown by the two horizontal lines, there is typically a link within the line discipline from the input queue to the output queue. If the line discipline is echoing the input character, this is done by moving a copy of the character that is input to the output queue. Also, if flow control is being done, when the stop character is input this has to stop the output side of the discipline. Similarly, when the start character is entered, the output side has to be notified.
The interaction that we have to be concerned with is if the interrupt key or the quit key is entered. Normally, entering either of these keys flushes both the input queue and output queue, in addition to terminating the process that is currently running. When a terminal discipline is in the raw mode, these two keys are no longer special and entering them won't flush the queues.

3.8 Flow Control

Most terminal line disciplines that are intended for interactive use, buffer characters in both directions. This allows a process to write data in chunk. Typically these chunks are either complete lines or buffers of some size. The driver accepts as much data as it can and then outputs it to the terminal as fast as possible .By doing it this way, the process that is producing the terminal output can continue executing while the line discipline and the terminal device driver output to the terminal device as fast as it can accept data. Terminal out put is usually limited by the connection speed-9600 baud, for example. If the output is being displayed on the terminal device too fast for you to read or comprehend ,you can stop the output by pressing a special character, termed the stop character .When you are ready to have more output displayed ,enter the start character. The stop character is typically Control-Sand the start character is typically control-Q.in addition to this out put buffering, terminal input is also buffered independ entry by the line discipline. By doing it this way, you can enter characters before a process is ready to read them. We can picture this as two some what independent queues within the terminal line discipline. As shown in the figure below. There is an input queue and an output queue for every terminal and pseudo-terminal.

3.9 Pseudo-terminal packet Mode

Flow control is best handled by the client system. If it is handled by the remote system, then when you enter the stop character, that character has to be transmitted across the network to the remote system, where the remote line discipline module will stop the output. But in time required to do this ,all the data that was already going from the remote system will have been displayed on your screen .The problem with doing flow control on the client system is that whenever the remote system is in a raw mode, the stop and start character cannot be interpreted as flow control.We need some way for the remote system's line discipline to notify the remote login server when the start and stop character are enabled or disabled.

There is another condition that we would like the client process to know as soon as possible .The terminal ioctl of TIOCFLUSH allows a processes to flush every thing that the line discipline has queed for input or output to the terminal device . If the process running on the remote system system issue this terminal ioctl, in addition to having the line discipline moduled on the remote system flush its output buffer , we would also like to flush every thing that is buffer across the network for the output on your terminial > This is another condition that the remote system 's line discipline modules knows about , which we would like the remote login server to also know .

To handle these cases , the 4.3BSD pseudo -terminial device driver supports an optional packet mode . This mode is enable by missing an ioctl of TIOCPKT to the pseudo -terminal master device , of certain events that happen in the slave 's line discipline moduled. When this mode is enable , every read of the pty master (by the remote login server in our example) returns either .

a single byte of zero . followed by the actual data from the pty slave. The first byte of zero is a flag byte indicating that the remainder of the buffer is normal data.
a single nonzero byte > The length returned by the read should be one. This byte is a control byte that indicates a condition that happened on the slave pty. The include file < iocth.h > contains the following definitions for this byte.

TIOCPKT_FLUSHHREAD

Indicates that the terminal 's input queue was flushed (i.e) all the characters on the read queue are flushed . for example . when you enter either interrupted key or quit key . This way . if you abort a running process with either of these keys , any out put that this process has already written or any input that you have entered, is discarded

TIOCPKT _STOP

Indicates that the terminal output has been stopped

TIOCPKT_START

Indicates that the terminal output has been transferred .

TIOCPKT_DOSTOP

Indicates that some thing has changed so that the terminal stop character is control _s and the start character is not Control _Q or the terminal is not in raw mode.

TIOCPKT_NONSTOP

Indicates that some thing has changed so that the terminal stop character is not Control_S or the start character is not Control _Q or the terminal is in raw mode.

Whenever one of the above control bytes is available to be read from the master

pty device the file descriptor for the master pty indicates that an exceptional conditions is present , if the select system call issued . This way the process that is reading from the master pty can differentiate between normal data and control information, before issuing the read .

The remote login server is only interested in three of these control indicators

TIOPKT_FLUSHWRITE

TIOPKT_NONSTOP

TIOPKT_DOSTOP

When the server reads any of these bytes from its master pty , it sends an out-of-band message to the remote olgin client . since the child remote login process is reading from the network, it arranges to reo receive the out-of-band notifications (the SIGURG signal and its actions are as follows

TIOCPKT FLUSHWRITE

Since the server's terminal output queue was flushed, the client should try to flush all the pending output that itcan.. It first issues an ioctl of TIOCFLUSHfor its standard ouput your terminal to discardany output data that is buffered in the line discipline. It then reads everything from the network, upto the out of band byte, and throws the date away. This way, any data that is buffered in the network, is also discarded. This is a good example of when we want the notification of out of band data to arrive as soon as possible. As soon as the receiver reads the out of band byte, it is going to throw away all the in band data that the out of band was sent ahead byte, it is going to in this case, the slave pty is no longer using control-Q and Contron-S for its start and stop character, or the slave pty is in the raw mode. In either case, the remote login client can no longer do flow control and the client must pass all characters through to the server process. This is what happens, for example, when you start the Vieditor on the remote system.

TIOCPKT-DOSTOP

Here the slave pty is not is raw mode and its start and stop characters are Control-Q and Control-S. This allows the remote login client to handle flow control on the local system.Tdo this, the client puts the line discipline module for your terminal in cbriak mode instead of raw mode to have dthe client do the flow control processing.An emampleof this notification is if you terminate the Vi editor, for example.

The notification provided by the packet mode of the pseudo-terminal is hard-wired for

having Control-Q and Control-Q and Control -S as the start and stop characters 4.3 BSD allows you to set the start and stop characters to anything you like. Yet packet mode bases its notification on these two specific values. The reason for this is that few people change their start and stop characters from their defaults of Control-Q and Control-S.BY assuming these values of these characters. This handles most cases. But as long as your start and stop characters on the remote system are not Control-Q and control-S the client will never do the flow control.

Now we can reexamine the description that we mentioned earlier the remote login facility is remote-echoed, locally flow controlled ,with proper back-flushing of output.

1. The remote-echo is because the client has the server do all the echoing.
2. we describe above how the flow control for the terminal output is done on the client s system,as long as the start and stop characters on the remote system are Control-Q and Control-S and if the remote line discipline is not in the raw mode .
3. We also described how the client; back flushes any pending terminal output, when it receives the flush notification from the ion from the server.

In addition to these three points, we can add that the remote facility also propagates changes in the clients, window site to the size to server

File Transfer

4. FILE TRANSFER

4.1 INTRODUCTION

File transfer is an important part of any network. The objectives of the file transfer are

1. to promote sharing of files computer programsand\ordata),
2. toencourage indirect or implicit (via programs)use of remote computer
3. to shield a user from variations in file storage systems among hosts, and
4. to transfer data reliably and efficiently.

4.1 file transfer packet formats

The RRQ and WRQ packets are sent by the client to the server to receive a file form the server(RRQ)or send a file to the server (WRQ).They specify the filename and its transfer mode. The file name and mode are both specified in ASCII. Both of these strings are variable length and terminated by a zero, which are shown explicitly in the figure above as EOS(end-of-string').The mode must be the string net ASCII or the string octet.l

A data packet contains the actual data bytes along with a block number.The length the data packet is between 0and 512bytes.If the length of data is between 0and 511bytes,that data packet is the final one, otherwise the length is 512and there is more data to be sent. The block number is used by the other end to acknowledge which packet was the most recently received valid data packet.
The error packet is sent when an error occurs and it contains both an error code and an

shown below Every packet begains with a 2-byte opcode.

	opcode	string	EOS	string	EOS
read request (RRQ)	01	filename	0	mode	
	2 bytes	n bytes	1 byte	n bytes	byte

	opcode	string	EOS	string	EOS
write request (WRQ)	01	filename	0	mode	0

	opcode		
data	03	blocks	data

	opcode	
acknowledgement (ACK)	04	blocks
	2bytes	2bytes

ERRCODE	DESCRIPTION
0 PRESENT	NOT DEFINED SEE THE ERRSTRING
1	FILE NOT FOUND
2	ACCESS VIOLATION
3	DISK FULL OR ALLOCATION EXCEED
4	ILLIGAL FILE TRANSFER OPERATION
5	UNKNOWN PORT
6	FILE ALREADY EXIST
7	NO SUCH USER

optional message string providing additional detail on the derror condition.The errstring is specified in ASCII and is variable length, terminated with a byte of zero. The error codes are given in the following figure4.2. The restring can be present with any errcode value, to provide additional information.

Figure 4.2.File transfer error codes.

All the 2-byte fields in these packets, the op code, block and err code, are stored in network byte order.

Let us show some examples of the packets that are exchanged between a client and server.

1.The client asks to receive a file from the server.

CLIENT (receiver)	SERVER (sender)
RRQ →	
	← data, block#1
ACK, block #1 →	
	← data, block#2
ACK, block#2 →	
	data, block # 3
ACK, block#3 →	

Here we use the term receiver to designate the end that is receiving the data packets, and the term sender for the end that is sending the data packets.

2.The client asks to send a file to the server.

```
CLIENT              SERVER
(sender)            (receiver)

WRQ          →
             ACK, block #1
data, block#1 →
             ACK, block#2
data, block#2 →
             ACK, block#3
data, block # 3 →
      (etc.)
```

Since error free channel is supplied by the transport layer ,we need be concerned about the handling of packets being lost. Most errors like network errors and other fatal errors cause termination of the program. Some errors like file not found , access violation for a WRQ or a RRQ request are just displayed on to the screen and program is not terminated.

Note that once the initial RRQ packet is sent by the client, the remainder of the file transfer procedure is symmetrical between the client and server .Both can send and receive data packets, acknowledgements, and error packets. We will take advantage of this fact in our implementation of the file transfer, to use as dsmush of the source code as possible in both client and server file transfer procedures

4.3 Security

Security is provided in our file transfer facility added to the remote login application. The user can access only files which access only files which have sufficient permission to do the required operation (i.e read permission for read operation and write permission for write operation).

In our application, when a user sends a get request, he must be having write permissions to write a file in the local directory and read permissions for the remote file the user wants. For a send request he must be having write permissions to write a file in the remote directory and read permissions for the local file the user wants to send.

Security added to the file transfer facility, is done as follows.

When a user requests for a get or send file request, a child process is created with user

iD and group iD of the local (remote)user in the cal(remote)user in the client(server)process. The child process will be having the permissions of the local (remote)user in local remote user in the local (remote)computer. Thus the child process can only access those files for which it is given permission. Thus, security is incorporated.

4.4 Data Formats

There are two formats of data transfer supported by our file transfer facility net ascii and octet. The net ascii format is used for transferring text files between the client and the server. The standard ASCII character set is used and the end of each text line is designated by a carriage return (octal 15)followed by a line feed (octal 12).if there is a carriage return in the text file, it is transferred as carriage return followed by a null byte (octal 0).The presence of a carriage return followed by any other character is undefined. By defining a standard format for the text file that is being transferred, it is possible to transfer data between two different systems, it is the responsibility of the client and the server to convert the local file representation to and from net ascii.

The octet data format is used for transferring binary files. We defined the term octet as being an 8-bit quantity, which we call a byte. There are two primary uses of the file transfer facility to transfer binary data between systems. First, two systems with same architecture can obviously exchange a binary file without any problems. Second, if a system receives a binary file and then returns it to the system that sent it originally, the format of the file must not change. This scenario can be used to provide a file server. The cafile server. The clients send their files to the server in binary made and retrieve them later in binary mode. The server would not be trying to interpret the contents of the binary file, it is merely storing it on its local file stem. As long as it uses the same storage technique to store and fetch a binary file ,the actual contents of the file won't change.

4.5 Client User Interface

In developing the implementation of the file transfer facility, a user interface is needed. The commands provided to an interactive user are described here.

Mode transfer-mode

Set the mode for file transfer .The transfer-mode must be either ascii (for a netascii transfer)or binary (for an octet transfer). The default file transfer mode is ascii.

binary

Set the mode for file transfer to binary (octet). This command is shorthand for mode binary.

ascii.

Set the mode for file transfer to ascii (net ascii).This command is shorthand for mode ascii

get remote name {local name}

Get a from the server. The remote name can be either the name of a file, in relative path name or absolute path name of the file. If the local name is not specified, then local filename is same as that of the remote name. If the remote file name is given as relative path name, then the present working directory of the remote login session is taken and prefixed to the remote name. The mode of the file transfer depends on the most previous mode command.

Put localname [renitebane]

Put a file from the server. The remote name can be either the name of a file, in relative path name or absolute path name of the file. If the remote name is not specified , then local filename is same as that of the local name. If the local file name is given as relative path name, then the present working directory of the remote login session is taken and prefixed to the remote name. The mode of the file transfer depends on the most previous mode command.

Status

Shows the current status of the file transfer program.

help

print a 1-iine summary of each fcommand.Equivalent to the command
print a 1-line summary of each command .Equivalent to the help command.

5. CONCLUSIONS

This project provides the combined features of both

1. RLOGIN
2. FTP

Thus it provides the user both the facilities of logging onto the remote system and transferring files between the local and remote systems.

But, our project provides this service to UNIX users only. So, this project can be extended to provide its service to all the users by replacing the RLOGLY with TELNET thus mixing the features of TELNET and FTP.

BIBLIOGRAPHY

1.UNIX NETWORKING PROGRAMMING

W Richard Stevens.

2 . COMPUTER NETWORKS

ANDREW S. Tanenbaum.

3. INTERNET WORKING WITH TCP/IP (VOL 1, 2, 3)

-Douglas E. Comer

Appendix

A User Manual & Guide

Remote Login With File Transfer facility ;is a utility that allows users on Unix to login onto a remote computer and also to transfer files between the local and remote computers at the same time. It also has a provision to execute local commands

A.1Getting Started

To run the utility, type in at the Unix prompt

slogin remote host -1 remote user name

Here remote host is the address or name of remote host and remote user name is the user name you want to login as, onto the remote computer. Then wait until the password (if any)is asked.

Enter the password of the remote user correctly. Then you will be logged on to the remote computer. Then wait until the password (and the prompt appears and you can work as if you have really logged at the remote computer.

A2 How to log out ?

You can loge at in two ways

Type exit at the command line prompt and you will be logged out This is the normal way of logging out.

You can type escape character followed by or data the starting of the command line prompt and you will be logged out. The difference between these two methods is .

In the former method, your request will be serviced at the remote system, while in the latter method it will be serviced at the local system.

A.3 About -command.

Type i.e escape character)and type any of the following character

d You are logged out.
you are logged out

z suspend remote login process.

y suspend remote login process, but all out put from remote system.

A.4 A But File Transfer

Type as the first character at the command line prompt can type any file transfer utility command . Or type help to list out all file transfer utility command.

you can files through get command and send files through put command.

B. About files in source code

1. Client. C

This is the client program of our application REMOTE LOGIC WITH FILE TRANSFER FACFILITY. This is the program to be compiled to get the executable file, to be run on the local computer. When drun, connection is made to the remote computer, the login and password are asked and then we are logged on to the remote computer. The connection establishment is done by the function remd 1( )present d in the file mdc

2. Server. C

This is the server program of our application REMOTE LOGIC WITH FILE TRANSFER UTILITY .This has to compiled to get the executable file, which is run as the server for our service. This is what gets connected, when the client program makes a request for connection.

3. DEFS.H

This contains the definitions for the client and server programs for the file transfer purpose .

4. FTPCLI.C

This file contains the function ( ) which is called by the client program when the users enters a file transfer utility command.

5. CMD.H

Header file for user command processing functions of the file transfer utility.

3. CMDSUBR.C

Contains functions for user command pr0cessing of the file transfer utility.

4. CMD.C

Contains one function for every user command of the file transfer utility.These functions are called bythe docmd function from the cmdsubr.c.

5. CMDGETPUT.C

Contains do get () and do put ()functions that do the file get and put processing.

6. FILE.C

This handles all the Unix file 1\0.This includes any required conversions between the file transfer formats, netascii and octet, and the Unix file format .

7. FSMC

This is the finite state machine that drives the file transfer f processing. when a user enters either a get or put command, the functions do fd get and do put that are present in the cmd get put. c send the first packet to the server. Then the sm loop ()the the sm loop () is called to do the rest of the file transfer processing. The fsm loop()function is also called by the server to do all its file transfer processing.

12. SENDRECV.C

This file contains all the functions that send and receive packets to peer process during the execution of the file transfer of the file transfer utility command. Some of these are for only the client or server, and some are used by both.
12INITVARS.C

This file declares all the global variables and initializes them, used by the files defs.h. ftpcli.c. cmd.h. cmdc. cmdsubr.c cmdgetput c, tsmc, sendrecv.nettcpc.

8. NETTEP.C

This file does ds all the network handling for connection used by the file transfer utility.

14. REMD H

This file contains the function rcmtd1() which makes two connections to the remote server, one is used for the purpose for the purpose of file transfer.

15RW.C

This file contains functions read ()which reads n bytes from a descriptor, written()which writes n bytes to a descriptor and readline ( )which reads a line from a descriptor.

16.TIME.C

This file contains timer routines used for printing the time taken for a file transfer.

SOURCE CODE

```
#define  sigmask (m)   (1<<((m)---1))
# endif

struct  winsize currwinsize  /* current size of window */

void   sigpipe_parent ( ):      /* our  signal handlers  */
void   sigwinch_parent ( ):
void   sigcld  _ parent ( ) :
void   siggurg _ parent ( ) :
void    sigusr_parent ( ):
void    sigurg_child ( ):

#define get_window_size(fd ,wp ) ioctl ( fd , tiocgwinsz , wp )

int  sockfd2=1 ;


main (argc,argv)
int  argc ;
char  **argv;
{
    char   *host,*cp;
    struct  sgttyb   ttyb;
    struct  passwd  *pwd;
    struct  servent     *sp;
    int   uid ,      options=0 , oldsigmask;
    int   on =1;
    if   ((host = strrchr(argv [ 0 ] , '/' )) ! =null ) host++;
    else   host=argv [ 0 ];
    argv++;argc--;
    if   (strcmp  (host,  "slogin" ) ==0 )
                  host=  argv++ ,argc ;

         strcpy ( hostname, host ) ;

another :
         if   (argc>0  && strcmp  (*argv , "-d")==0 )
         {

/*
* Turn  on the debug  option  for the socket.
*/

argv++;argc--;
options : =so_debug;
goto  another;
}
if  (argc>0   && strcmp  (*argv , "-1 ") ==0)
{
/*
* specify  the server -user-name , insted  of using  the
*name of the person invoking us
*/
```

```
argv ++; argc --;
if  ( argc ==0 ) goto  usage ;
name = *argv ++;argc--;
goto another;
}
if  (argc>0  && strncmp (*argv, “-e” ,2 ) ==0 )
{
/*
*specify  an escape  character ,  insted of the  defult  tilde.
*/

escchar=argv [ 0 ] [ 2 ];
argv++; argc--;
goto  another;
}
if  (argc>0  SS  strcmp (*argv , “-8” )==0 )
{
/*
* 8-bit  input . Specifying  this forces us to use  RAW mode
* input  from the  user's   terminal. Also, in this  mode we
* won't  perform  any local flow control.
*/

          eight = 1;
          argv ++


if  (argc >0 && strcmp ( *argv , “--L “ )== 0)
{
/*
*  8-bit   output.   Cause  us to set  the LLITOUT  flag,
*   which tells  the line  dicipline :  no out put   translations .
*/

litput +1;
argv++; argc --;
go to another ;
}
 if   ( host ==NULL )   goto  usage;
if    ( argc>0 ) goto usage;    /* too many  command  line arguments  */
/*
*  GET the name of the user  invoking  us : the client -user-name
*/
if  (( pwd=getpwuid ( ) ) )   ==  NULL )
{
fputs ( “ Who aer you ?\n”,stderr );
exit ( 1 );
}

/*
*  Get the name  of the  server  we connect to.
*/

if  ( ( sp=get serverbyname(“slogin”, “tcp”) ) == NULL)
```

```
f puts (" slogin /tcp: unknown service \n" , stderr);
exit ( 2 ) ;
}
/* Get the name of the terminal from the enviroment .
*also get the terminal 's speed . Both the name and
* the speed are passed tothe server as the " cmd"
* the argument of the rcmdl ( ) fuction . this is some thing
* like " vt100/9600"
*/
if ( ( cp=getevn ("TERM") ) ! = NULL ) strcpy ( term, cp);
if (ioct 1 (0, TIOCGETP, & ttyb)==0)
{
strcat(term,"/");
strcat (term, speeds[ttyb.sg_ospeed] );
}
get_window_size ( 0, &currwinsize );
signal ( SIGPIPE , &sigpipe_parent );
printf ("nclient. c: before rcmdl\n" );
/*
* Block the SIGURG and SIGUSRI signals . These will be handled
*by the parent and the child after the fork.
*/


oldsimask=sigblock ( sigmask ( SIGURG ) : sigmask ( SIGUSRI ) );
/*
* Use rcmd 1 ( ) to connect the server . We pass the terminal -type/ *
*speed as the "command argument , but the slogin server takes the
* the command argument from the connection established by rcmd1( ) and
*passes it to the login shell executed by it .
*sockfd is the socket for the connection to the slogin server .
* for the porouse of the remote login .
* Also , rcmd 1( ) makes another connection to the server for the
* porpouse of the file transfer between the remote and local m/cs
* and sock fd2 is the socket for that connection
*/

sockfd =rcmd1(&host , sp->s_port, pwd-> pw_name, name ?
                    name : pwd->pw_name, term , (int * )&sockfd2 );
if (sockfd<0)
exit ( 1 ) ;
if ( ( options & SO_ DEBUG) &&
SETSOCKOPT (SOCKFD, sol_SOCKET, SO_DEBUG,on , sizeof ( on) < 0 )
perror (" solgin : setsockopt ( SO_ DEBUG ) " }:
/*
* Now change to the real user Id and real group ID.
* we have to get the priviledge port that rcmd 1( ) uses ,
* however we now want to run as the real user who invoked us.
*/
if ( setgid (getgid( ) ) < 0 )
{
perror ( " slogin : setgid ");
exit ( 1);
}
```

```
uid= getuid ( ) ;
if ( setuid ( uid ) < 0 )
{
perror ( "slogin : =setuid " );
exit ( 1 ) ;
}
doit ( oldsigmask );
   /* NOTRECHED*/
usage :

fputs ( " usage : slogin host [-ex ] [ -1 ] [usersname ]        [ -8 ] [ -L ]\n"
,stderr );
exit ( 1 ) ;
}
int childpid;
/*
 * tty flags.
*/
int defflags;          /*  the sg_flags word from the sgttyb struct */
int tabflag;          /* the two tab bits from the sg_ flag word */
int deflflags;
char deferase;        /* client's erase   charactermmm*/
char defkill;         /* cliient's kill character */
struct tchar defcts;
struct itchas defits;

/*
 * If you set one of the special terminal characters to-1 , that effectly
 * disables the line discipline from the processing that special character .
 * Weinitialize the following two structures to do this. However,the code
 *replases the -1 enttries for stop the output and start-output with the
 * actual values of these two characters ( such as ^q/^s ). this way , we
 * can use cbreak mode but only have the line discipline do flow control .
 * all other special characters are ignored by our end and passed to the
 *server 's line discipline .
 */
struct tchars notc={-1,-1,-1,-1,-1,-1};
/*disables all the tchars : interrupt , quit , stop, --output start -output,edf */
struct ltchars noltc= {-1,-1,-1,-1,-1,-1};
        /* disables all ltchars : suspend , delayed --suspend ,
        reprint -line , flush, word-erase, literal-next */
        doit (oldsigmask)int oldsigmask; /* mask of blocked signals */
        struct sgttyb sb ;

ioctl ( 0, TIOCGETP , ( char * ) &sb );   /* get the basic modes */
defflags=sb.sg_flags;
tabflag=defflags &       TBDELAY;       /* save the 2 tabs bits */
defflags &=ECHO : CRMOD;
deferase=sb.sg_erase;
defkjill=sg_kill;

ioctl ( 0, TIOCLGET, ( char * ) &deflflags );
ioctl (0,TIOCLGET, , (char *)&deftc);
notc.t_startc.t_start;              /* replace -1 with one char */
```

```
notc.t_stopc=deftc.t_stopc;          /* replace -1 with stop char */

ioctl(0,TIOCGL TC, (char *)&defltc) ;
signal ( SIGNT, SIG_IGN);
setsignal (SIGHUP,exit ) ;  /*HUP or QUIT go straight to exit ( ) */
set signal (SIGQUIT,exit ) ;
if  (( childpid  = fork ( ) ) <0 )
{
        perror( "SLOGIN : fork" );
        done (1);
}
if   (childpid==0)          /* child process == reader */
{
        ttty_mode (1) ;
if  (reader(oldsigmask)==0)
{
/*
*If reader ( ) returns 0, the socket to the server returned an EOF ,
* meaning the client logged out of the remote system .
*/

prf ("Connection closed " );
exit ( 0 );
}
/*
* If the reader ( ) returns nonzero , the socket to the server returned an error . Something
* went wrong
*/
sleep  ( 1) ;
prf ( "\007 child : connection closed .");
exit ( 3 );
}
/*
* parent process == writer
*
*We still own the socket , and may have a pending SIGURG ( or might recive
*one soon) that            really want to send to the reader . Set atrap that copies such
*signals to the child once the two signals handlers are installed ,
*reset the signal mask to what it was before the fork .
*/

signal  (SIGCLD, sigcld_parent );
signal (SIGUSR1, &sigusr1_parent);
sigsetmask ( oldsigmask);       /*reenables SIGURG and SIGUSR! */
signal (SIGCLD, &sigcld_parent);
writer ( );

/* If the writer returns , it means the user entered "~." on the terminal or an
*error ocured during file transfer in this case we terminate and the server will
*eventually get an EOF on its end of the net work        connection .This should
* cause the server to log you out  on the remote system
*/

prf ("Closed connection . ");
```

```
done (0);
}

/*
*Enables  a signal handler , unless the signal is alreaddy  being  ignored .
*This function is called  before  the fork ( ), for SIGHUP  and SIGQUIT.
*/

setsignal (sig,, action )
int  sig;
void  (*action) ( );
{
register  int  omask ;
omask = ( sigblock ( sigmask  ( sig ) ) );
if  ( signal  ( sig ,action ) == SIG_ IGN )
                    signal ( sig,SIG_ING);
        sigsetmask ( omask);
}
/* we send  the child  a SIGKILL signal , which it  can't ignore , then
* wait  for it  to terminate .
*/
done  ( status )
int status ;
{
        int  w;
ttty_mode (0);
if ( childpid  > 0 )
{
signal ( SIGCLD , SIG_DFL );
if   ( kill ( childpid , SIGKILL)  >=0)
                    while   (( w = wait (0) )   >0 && w!= childpid);
        }
exit ( status ) ;
}

/*  Copy  SIGURGs to  the child  process.
*   the parent  shouldn't  get any  SIGURGs,  but if it  dose . just pass
*   them the child , as  it"s the child  that handles  the  the  out -of -band
*   data from the server .
*/

void  sigurg _parent ( )
{
        kill( childpid , SIGURG ) ;
}

/*
* Frist time . Send  the initial winndow sizes  to the server
* the  TIOCPKT_WINDOW  indicator  from the server . This  tells the
* client  to the enable  the  in-band  window -changing  protocol.
*/

void  sigusr1_parent  ( )
{
```

```
if  ( dosingwich==0 )                          /* frist time */
{
        /*
        *frist time . send  the initial window size  to the server
        * and enable  the SINGWINCH signal , so that  we pick up
        *any changes  from this point on
        */

send window ( );
signal ( SIGWINCH,& sigwinch_parent );
dosigwinch=1;
        }

/*
*   SIGCLD signal   handler  in parent .
*/

void  sigcld_parent ( )
{
        signal ( SIGPIPE, SIG_IGN);
prf ("\007client : sigpipe _parent :connection closed ");
done ( 1 );
}

/*
*writer main loop : copy  standrad  input ( user 's terminal ) to network .the
*standrad  input  is in the raw mode , however , we look  for four special sequense
* of characters:
*       ~.      terminate;
*       ~^D     terminate;
*       ~^Z     suspend slogin process;
*       ~^Y     suspend slogin process; but leave reader alone ;
*       !       file tranfer commands  and execution  of local commands ;
*
*This  handling  of escape  sequences ist't perfect .however  For  example , use
* slogin , then turn the vieditor  on the remote  system .
*Enter return , then  tiled ( vi convert-case-of-character command).
*then dot(vi's redo last command ) and you are logged out.
*/

writer ( )
{
        char c;
        register n;
        register bol =1;    /* beginning of line  */
        regester local=0;
        for ( ;; )
        {
n=read (0,&c,1 );
if  (n<=0)
{
if          (n  <  0  && errno==ENTER)
                continue;
                break;
```

```
}
/*
*If we are at the beginning of the line and recongnize the escape character ,
* then we echo the next character locally . If the commannd character is
* doubled , for ex If you enter ~~. at the biginning of the line , nothing is
* echoed locally and ~. is sent to the server if you entered ! at the biginning
*of the line , then file transfer and local commands execution     starts .
*/

if (bol)
{

        bol =0;
        if (c==escchar )
{
        local 1=1;          /* local echo next char */
        continue ;          /*next iteration of for -loop
        }
        else if ( c=='!')
{
        char ch;
        int chipd;
        int status ;
        /* restore user 'sterminal mode */
ttty_mode ( 0 );
write ( 1, & c,1);
signal ( SIGCLD, SIG _DFL);
/*
* fork a child process the
* user 's file transfer or local commands
*/
        if (( chpid = fork ( ) )  < 0 )
                return ;
if ( chpid ==0 )
{
        ftploop( stdin ) ;
exit ( 0 );
}
        status =1;
        wait (& status ) ;
        signal ( SIGCLD,&sigcld_parent );
        /*
        *If exitstatus of the child is 4,
        * an error has occerd during file
        *transfer and we have to exit ,
        *other wise we continue.
        */
        if (WEXITSTATUS(status)==4) return ;
        ttty_moce ( 1 );
        ch='\n' ;
        write ( sockfd, &ch , 1);
        continue;
        }
}
```

```
else if ( local ==1)
{
/*
* The previous char ( the first char of a line )
* was the escape char . look at the second char
* of the line and determinen if some thing
*special should happen .
*/
local =0;
if (c==‘ . ‘ : : c==deftc.t_eofc )
{

/*
* A ~. or ~ edf terminates
* the prent .Echo the period or EDF
*then stop.
*/
echo ( c ) ;
break;
}
if ( c==defltc .t _suspc : : C==deflltc.t_dsuspc)
{
        /*
        *a tilde -^Z or tilde -^Y stops the
        * parent process .
        */
        bol = 1 ;
        ehoc ( c ) ;
        stop ( c ) ; /* returns only when we are continued */
        continue;
}


/* If the in put was tilde -some other character ,
*then we have to write both the tilde and the
* other char to the network .
*/

if ( c ! =escchar )
if ( write ( sockfd , & escchar ,1 ) ! =1)
{
        prf ( “ line gone “) ;
break ;
}
}

if (write ( socked fd, &c, 1) != 1)
{
prf ( “ line gone “ );
break ;
}

/* Set a flag if by loking at the current character
* we think the next char is going to be the first
```

```
*/
bol= (c==defkill ) : :
      (c==deftc.t _enofc ) : :
      (c==deftc.t _intrc ) : :
      (c==defltc .t _suspc ) : :
      (c==' \r' ) : :
      (c=='\n');

         }
}
/* Echo acharacter on the standard output ( the user 's terminal ).
* This is called only by the writer ( ) function above to the handle the
* escape character s that we echo .
*/
echo ( c)
register char c;
char buf [8];
register char * p + buf ;
*p++ = escchar ;              /* print the escape char first */
C &= 0177;
if ( c < 040) {

/*
* Echo ASCII control character as a caret , followed
* by the upper case character
*/
* p++ = '^' ;
*p++ = c + ' @ ' ;
}

else if ( c == 0177 )  / * ASCII DEL character */
{
        *p++ = ' ^ ' ;
        *p++ = ' ? ';
  }
        else *P++ = c;
        *p++ = '\r' ;   /* need a returnn _linefeed, since it 's in
                        * raw mode.
                        */
        *p++ = '\n' ;
        write (1, buf, p_buf );


/* Stop the parent process ( job control ) . If the character entered by the
 *user is the "stopprocess"(^z) character , then we send the SIGTSTP signal
 *to both our self and the reader ( all the processes in the sending processes
*processes group )When this happens any thing sent by the server to us will
* 8be buffered by the network until the reader starts up again and reads it .
*however , if the character is the " delayed stop process " (^y) character . then
*we stop only ourself and not the reader . this way the reader continues output
*ing any data that it recives from the server.       stop (cmdc )
*char cmdc;
{
```

```
                    ttty_mode ( 0 );        /* Frist reset the terminal mode to normal */
                    signal (SIGCLD , SIG_ING); /* ignore SIGCLD in case of child stops too
*/          kill (cmdc  == defltc.t_suspc ) ?  0 : getpid ( ) , SIGTSTP )
        /*
        * SIGWINCH signal handler.
        * we are also called above . after we've ressumd after being stopped .
        *We only send a window size message to the server if the size has changed
        *note that we used the flag " dosigwinch" to indicate if the server supports our
        *window_size_change protocol . If the server dose not tell us that it supports it
        *( see sigusr1_ parent ( )
        */
                struct winsize ws;

                if (dosigwinch && ( get_window_size ( 0,&wsk) == 0 ) &&(bcmp((char * )
                        &currwinsize,size of ( struct winsize )) ! = 0 ))

        {
                currwinsize=ws ; /* store new size for next time */
                send window ( );  /* and tell the server */
            }

        }

        /* send the window size to the server via the magic escape . Note that we
         * send the 4 unassigned shorts in the structure in network. byte order , as it
         *is possible to be running the client and the server on system with different
         * byte orders ( avax and asun , for example ).
         */
        send window ( )
        {
                char obuf [4+sizeof (struct winsize )];
                register struct winsize *wp;

                wp=  (struct winsize * ) (obuf+4);

                obuf [0]=0377  ;  /* these 4 bytes are the magic sequences */
                obuf [1] =0377;
                obuf [2]= 's' ;
                obuf [3]= 's';


        wp->ws_row =  htons ( currwinsize-ws_row );
        wp->ws_col = htons (currwinsize.ws_col);
        wp->ws_xpixel =htons ( currwinsize .ws_xpixel );
        wp->ws_ypixel  = htons ( currwinsize .ws_ypixel );

        writer ( socked , obuf, sizeof (obuf ));

        /*
        * Reader main loop: copy network to standrad output ( user's terminal)
        */

        char rcvbuf [8*1024];        /* read into here from the network */
        int rcvcnt ;                 /* amount of data in rvcbuf [ ] */
```

```
int rcvstate              */ READING OR WRITING: cossigurg_child knows whether
                               aread  or write  system  call was interrupted .*/

int parentipd ;           /* parent  PDI  , from the fork   */
imp_buf  rcvtop ;           /* set jump /long jump  buffer  */
# define  READING 1        /*   values for  rcvstate  */
reader (oldmask )

int  oldsigmask ;                       /*  signal  mask from  the parent  */

{

        int  pid    =  getpid  ( )  ;
        int n, renameining;
        char    * bufp=rcvbuf ;
        signal     ( SIGTTOU ,SIG_ ING );
        signal (SIGURG, &sigurg _child );
                                  /* out _of _band  data  from the server    */
        fcnt1   ( sock fd,f _SETOWN , pid );
                                  /* to recive  SIGURG  signals  at bigning */


        parentpid=getppid  ( ) ;  /* see the long jump  in sigurg _child ( )   */

        sigsetmask (old sigmask );     reset signal mask*/


for    (;;)
{
        /*
        * Reader main  loop  --  read as  we can  form the network  and write it to the
standrad                          noutput
        */


While  (( remaning + rcvcnt --(bufp  --rcvbuf  ))  >  0 )

/*
*while there is a data in the buffer to write
*write it to the standrad output
*/

{
        rcvstate =WARNING;
        if  (( n=write  (1, bufp , renaming  )) < 0 )
{
        if  (error  ! =EINTR  )    return    -1;
        continue ;
}
        bufp=n; /* incr  pointer  past what we wrote */
}
```

```
/*
* There is nothing  in our  buffer  to weite , so read , so  read from the net work
*/
bufp = cvbuf ;      /* ptr  to start  of  buffer */
rcvcnt =0 ;         /* # bytes  in the buffer */
rcvstate  =reading ;
rcvcnt =read ( sockfed , rcvdnt ==0) return  /*  user  logged  out from  the remote system   */
if ( rcvcnt < 0 )
{
     if ( error == EINTR ) continue ;
      perror ("read ") ;
          return   -1;
}

/*
*  This is the  SIGURG signal handler  in the child . here we process the  out-of-
*  bamd  signals  that arrive from the server .
*/


viod sigurg_child ( )

          int flushflag , atoobmark,n,rcvd ;
          char   waste [BUFSIZ], ctlbyte;
          struct sgttyb  sb;

          rcvd =0;
while (recv (sockfd  , &ctlbyte ,1,msg_oob)   <0  )

{
          switch (errno )
case EWOULDBLOCK:

/*
*   The urgrnt data  is not here  yet.
*  it may not  bepossible  to send  it
* yet if you are blocked  for output   and our input  buffer is full . first
* try to read as much as the tecive  buffer has a room for .
*Note  that nither  of reads below will go past  the oob mark .
*/

if ( rcvcnt   < size of (rcvbuf ))

{
          n=read (sockfd , rcvbuf+rcvcnt,
          sizeof (rcvbuf )-rcvcnt;
          rcvd  +=n;

/* rember how much we read  */
*we have  no choice  but to read in to the waste basket
*/

          n= read  (socked , waste ,size of (waste));
if  (n  < +0) return
```

```
}
continue ;
/* try to read  oob byte again */
default :
        return ;
        }
}
/* note that  in the TICOPKT mode , any number  of control  bites
*may be  on the control  byte  . so we have  to test  for all the
*ones  we reintrested  in .
*/

if (  ctlbyte & TIOCPKT_WINDOW)

{

/*  We  get this  control byte  from the server  after it has  started . It means  that
* the server  is started  and it needs  to know  the current  window size . We sent
* the  window  size to the server .
*/

kill ( parent pid, SIGUSR!);

}
if  (! eight  && ( ctlbyte &  TIOCPKT_NONSTOP ))
{

/*  Either  the server   is not using  ^S/^Q  or the server  is  in raw mode . We
* must set  the user's terminal to raw  mode . This  disables  flow control  the
*client  system .
*/

ioictl (0,TIOCGRTRETP, ( character  * & sb );
sb .sg _flags  & =  ~ cbreak ;  /* Cbreak off */
sb.sg _flags    : RAW :   /* RAW ON   */
not.c t_stopc  = deftc.t_ stopc;      /*    enable stop  */
notc.t_startc   = deftc.t_startc;     /* enable start      */

iotc1  (0, TIOCESETC, (char  * ) &noct );
}

if  ( ctlbyte &  TIOCPKT _ FLUSHWRITE )
{

/* The terminal  output  queue  on the server  was flushed . Frist we
* flush our terminal  output  queue  (  the output  queue  (  the out put queue  for the
terminal*)

        flushflag  = FWRITE ;
        ioctl (1, TIOCFLUSH, ( char  * ) & flushflag  );
        for (;;)
{

        if   ( ioctl (sockfd , SIOCATMARK , & atoobmark )   <  0)
```

```
{
        perror (" ioctl SIOCATMARK  error ");
        break ;

}
if   ( atoobmark  )
        break ;
    if  ( ( n=read (sockfd , waste , suize of ( waste ) ))   < =  0 )
                break ;
}

/*
*  We  do not want any pending data  that ew  have already read in to  the reciver
*buffered to be out put so clear  the reciver  buffer  ( i.e  just  set cvcnt  = 0 ).  also ,
* if we  were  hanging  on a write to standrad output When interrupted , we dont want
*it  to  restart  , so we long jump  back  to ht etop  of the loop  if we were  reading , we
*want  to restart it any way.
*/

rcvcnt  = 0;

longjump ( rcvtop , 1);                                     / * back to the setjump   */
                                                        /* the  arg  of  1 is not  used  */
}
/*
* if we read data the recive  buffer  above  ( so that we could  read the  00b  byte )
*and if we were interrupted  during  aread , then  longjmp  to the loop  to the write
* the  data  that was  recived.don't abort  a pending  write,  however ,
*/

if ( rcvd  > 0 &&  rcvstate  ==READING )
        longjmp (rcvtop,1 )
return ;   /* from  the signal handler  ; probally  causes  an  EINTER */
}

/*  Set the terminal  mode . this fuction  affects   the user's terminal
* We are called   by both  the parent  and child .
*./

ttty_mode(mode)
int  mode ;   /* 0-> reset  to normal ;  1->  set for login  */
{

        struct  tchars  * tcptr ;
        struct itchars   * ltcptr
        struct sgttyb sb;                    /* local  modes  */
        int  1 flags ;                  /* local mode word */
ioctl  (0, TIOCGETP , ( char  * ) & sb )
 ioctl  (0,  TIOCLGET , ( char  * ) & lflags );
switch   ( mode)
{

        case    0:
                    /*
```

```
                    * This is to reset the terminal to how it is found before .
                    */

                    sb.sg_flafs &= ~(CBREAK : RAW : TBDELAY );
                    sb.sg _flags : = defflags : tabflag;
                    tctpr= &deftc;
                    ltcptr = &defltc ;
                    sb.sg_kill=defkill;
                    sb.sg_erase=deferase
                    lflags=deflflags;
                    break;
        case 1:
        /* to set terminal to raw mode .we default to CBREAK mode unless the -8
         * flag is specified in which case we have to use the raw mode .
         */

        sb.sg_flags := (eight ? RAW : CBREAK );
        sb.sg _flags &= ~defflags;
if (( sb.sg_flags & TBDELAY) ==XTABS )
        sb.sg_flags & =~TBDELAY ;
        tcptr = &noct;
        ltcptr = & noltc ;
        sb.sg_kill = -1;
        sb.sg_erase = -1;
        if (litout)
                iflags :=llitout;
                break;
                default:
return;
}
ioctl (0, TISCOLTC, (char * ) ltcptr );
ioctl (0, TIOCSCETC, char *)tcptr );
ioctl (0, TIOSCTEN, (char *) &sb );
ioctl (0, TIOCLSET,(char *) &lflags);
}
/*
*  Fatal error.
*/ prf (str)char
* str ;
{
                fputs (str, stderr );
        fputs ("\r\n", stderr ); /* return & new line , in case raw mode */
```

```
/**********************************************************************************************
                                   SERVER.C
**********************************************************************************************/
        /*
        *slogin server : the following data is sent across the network connection by the rcmd 1( )
                *function that the slogin client uses:
        *secondary connection port number ,
        *client user's name ,
        *server user's name ,
        *terminal type /speed ,
        *data.
        */

        # define SERVER

        #include < stdio.h>
        #include < sys/param.h>
        #include < sys/stat.h>
        #include < sys/socket.h>
        #include < sys/wait.h>
        #include < sys/file.h>
        #include <sys/file.h>
        #include < sys/types.h>
        #include < sys/time.h>
        #include < netinet/in.h.>
        #include < pwd.h>
        #include < signal.h>
        #include < bsd/sgtty.h>
        #include < stdio.h >
        #include < netbd .h>
        #include < syslog.h>
        #include < strings.h>
        #include < errno.h.>
        #include " file .c"
        #include " fsm.c"
        #include " invitators.c"
        #include " sendrecv.c"
        #include " nettcp.c"

        extern int errorno;

        /*
        * We send aTIOCPKT_WINDOW notification to the client when we start up .
        *This tells the client that we support the window size change protocol . The
        * value for this (0x80) can't over lap the kernel defined TIOCKPT_xxx values.
        */

        #defind TIOCPKT _WINDOW
        #defind TIOCKPT_ WINDOW 0x80
        #define
        char * env [2]; /* the enviroment we built */
        static char term [64]=" TERM=";
        #define ENVSIZE ( sizeof ("TERM=")-1)
        # define NMAX 30
```

```
char cliuname [nmax+1]; /* user name on the client ' host */
char servername[NMAX+1]; /* user's name on the server host */
char        *dir _name;
int keepalive =1; /* set to 0 with -n flag *
#define SUPERUSER ( pwd) (( pwd) ->pw_pwuid == 0)

int reapchild ( );
struct passwd * malloc ( ) ;
int one =1;        /* for set socked opt ( ) */
int sockfd 2:      /* socked for the file transfer connection */

main ( argc , argv )
int argc ;
char ** argv;
{
extern int opterr , optiond;
int ch, addrlen ;
struct socaddr _in cli_addr;

openlog ("slogind " , LOG_PID :LOG_COSN, LOG_AUTH );
opterr=0;
while ((ch gettopt (argc,argv, "ln" )) !=EOF )
switch (ch)
{

case '1' :
{
extern int _check_rhosts_file ;_check_rhosts_file=0;
}

break ;
case 'n'  : /* dont enable SO_KEEPALIVE*/
                      keepalive =0;
                      break;
              case '?' :
              default :

                      syslog (LOG_ERR,
                                  "usage: slogin [-1] [-n]");
                                  break;
                      }

       argc -= optind;
       argv +=optind;

/*
* we assume we are invoked by intend , so that the connection is on , is
*open on description 0,1 and 2 get the inter net address of the process to
*perform authentication checking.
*/

addrlen =size of (cli_addr);
if (getpeername (0,(struct socked dr *)&cli_addr, &addrlen ) < 0)
```

```
{
                    syslog (LOG_ERR,
                              "couldn't get peer name  of remote  host : %m");
                              fatal perror("cant get peer name of host ");
}

if  (keepalive && setsockedpt (0
                                        SOL_SOCKET,
                                        SO_KEEPALIVE,
                    (char * ) &one , size  of (one )) < 0)
syslog (LOG_WARNING, "setsocked ( SO_KEEPALIVE): %m");
          do_it (&cli_addr) ;

          int child:
void cleanup( ) ;
char line [11];
extern   char *  inet_ ntoa ( );
struct winsize win ={ 0,0,0,0,};

do_it (cli_addrp)
struct  sockeddr _in  * cli_addrp; /* client's internet  address   */
{

          int  I, masterfd, slavefd, chilpid;
          int  authenticated =0, hostok =0;
          char  remothost [2*MAXHOSTNAMELEN  +  1];
          regester   struct  hostent  * hp;
          struct hostent hostent ;
          char  c;
          short  clisecport ;
          int cc, oursecport

/*   try to lok  up the  client's  name , given  its internet  address , since  we  use the  name
for the   *authentiction .
*/

cli _ addrp-> sin _port =nthos (( unsigned  short ) cli_ addrp-> sin_port );
hp= gethostbyaddr (( const char *) &cli_addrp->sin_addr,
                    sizeof (struct  in_addr),
                    cli_addrp->sin_family);
          if  (hp==NULL)
{
          /*
          *  Couldn't  find  the client's name .
          *  use its  dotted_decimal  addess as its name .
          */
hp=&hostent;
hp->_name =inet_ntoa (cli_addrpaddrp-> sin_addr);
hostok++;
}
else
{
hostok++;                              /* to be written  afterwards */
```

```
/*
* Verify that the client's address is an internet address and that is an internet
* addess and that it was bound to a reserved port.
*/

if ( cli_addrp-> sin family ! =AF_inet ::
    cli_addrp-> SIN_PORT >=IPPORT_RESERVED ::
cli_addrp-> sin_port < IPPORT_RESERVED /2)
{
syslog (LOG_NOTICE
                    " connection from %s on illegal port ",
                    inet_ntoa(cli_addrp-> sin _addr));
            fatal (0, "permission denied");
}

/*
* read the secondary port no . at which the client is listening and connect to that port .
and     *esttablished a connection for file transfer .
*/
 alarm (60);
clisecport =0;

 for (;;)
        {

if (( cc = read (0, &c,1)) ! = 1)
{
        if ( CC < 0)
                syslog (LOG_NOTIC, " read : %m" );
                shutdown (0, 2);
                exit (1) ;
}
 if ( c==0 )
    break ;
clisecport = (cliseport * 10 ) + ( c-'0');
}
alarm (0);

if (clisecport ! =0)
{

                if (clisecport >= IPPORT_RESERVED )
{
SYSLOG ( LOG_ERR, "2nd port not reserved ");
exit (1);
}

/*
* Write null byte back to the client telling it that everything is o.k.
*/

write (0, "",1);
if (do_rlogin (hp->H_name)==0)
{
```

```
if (hostok)

authenticated ++;
else
write(0, "slogin : Host address mismatch. \r\n", size of ("slogind: Host addess mismatch.
\r\n")-1)}
/*
* Alloocated and open amaster pseudo_terminal.
*/

for (c='p';c<='q';c++)
{

struct start statbuff; structpy (line , "\dev\ptyxy ");

line [8]=c;
line [9]='0' ;
if (start (line , &statbuff) <0 ) break ;
for ( i=0;i<16;i++)

{
        line [9]="0123456789abcdef" [I];
        if (( masterfd +open (line ,0_RWDR)) > 0)
                        goto gotpty
}
}fatal (0, " Out of ptys ");
/* NOTREACHED */

gotpty:
/* set window size all to 0*/ ioctl (masterfd , TIOCSWINSZ, & win );*/

/*
*Now open the slave pseudo _terminal corresponding to the master that opned above
*/


line [5]='t'; /* change "/ dev /pty/xy " to "dev /ttyxy " */
if ((slaved =open (line ,0_RDWR) )<0)
        fatal perror (0, line);
if (fchmod (slavedfd ,0) )
fatal perror (0, line );
/*
*Now reopen the slave pseudo -terminal again and set it 's mode . This
* gives us a clean control terminal .
*/
line [5]='t';                    /* change " / dev /ptyxy " to "/dev /ttyxy */ptyXY */
if ((slaved fd =open (line ,0_RWDR )) <0)
        fatal perror (0,line);
if (fchmod (slavefd ,0 ))
fatal perror (0,line );
signal (SIGHUP<SIN_ING );
vhangup ( );
signal (SIGHUP,SIG_DFL);
```

```
/*
* Now  reopen  the slave  pseudo _terminal  again  and set  it's mode . this gives
* us a clean control
* terminal .
*/

if  ((slavefd =open (line , 0_RWDR ))  <0)
        fatalperror (0,line);

setup_term (slavefd );
#ifdef DEBUG
{
        int       tt;
        if ((tt=open ("/dev/tty", 0_RWDR ))>0)
        {

        ioctl(tt, TIOCNOTTY<)0);
                close(tt);
        }
}

#endif
        if (( childpid=fork ( ) ) < 0  )

        fatal perror (0," " ) ;
        if  (childpid ==0)
{
/*
 *  Child  process. Become  the login  shell  for the user .
 */

close  ( 0 ) ;                                  /* close socket  */
close (master fd );                             /* close  pty  master  */

 close ( sockfd 2);                          /*  close secondary  connection socket  */

dup2 (slavefd, 0 ) ;  /* pty  slave  is 0,1,2, of login  shell  */
dup2  (slavefd,1 )  ;
duf 2  (slavefd,2)   ;
close (slavefd) ;

/*
* invoked  the /bin/login  with _p and _h options ._P flag  tells  login  not to distroy the
enviroment .
*  -h flag  passes the name  of the client 's  system  to login , So it can be placed  in the utmp
and       wtmp
* entries
*/

execl ( " /bin/login  ", " login " , "-p" ,  "-h"  , hp -. _name ,
servuname,  ( char  * ) 0 );

        fatalperror  (2, "/bin/login" );
```

```
        /* NOTREACHED */
}

/*
* parent process .
*/

closed ( slavedfd ) ; /* close slave pty , child uses it */
ioct1 (0, FIONBIO, &one ) ; /* nonblocking i/o for socket */
ioctl (master FIONBIO , &one ); /* BSD pty packet mode */

signal (SIGTSTP, SIG_ING);
signal (SIGCLD , &cleanup );
setpgrp (0,0);   /* set out our process group to 0 */
  protocol (0, masterfd, socked2 );
signal (SIGCLD,SIG_IGN);
cleanup ( );
}


# define pkcontrol (c) (( c)& (TICOPKT _FLUSHWRITE :\
                TIOPKT_NONSTOP: TIOCPKT_DOSTOP))

/*
* The following byte always gets along with the pty packet mode control
* byte to the client .It 's initialized to TIOCPKT_WINDOW but this bits gets
* turned off after the client has sent the first
* window size . there after this byte is 0.
*/

char oobdata [ ] ={ TIOCPKT_WINDOW };
char magic [2]={0377,0377}; /* in-band magic byte */

int control (pty,cp,n,)
int pty; /* fd of pty master */
char * cp ;    /* pointer to first two bytes of control sequence */
int n;

{
struct winsize w;
if ( n< 4+ sizeof (w) :: cp [2] !='s' :: cp [3] ! ='s')
        return 0;


/*
* once we recive one of these in_band control request from the client
* we know that itrecived the TIOCPKT _WINDOW message that we sent
* it on start up. We only send this control byte at the begning totell the client
* that we support window support window size changes. now we can turn off the
* TIOCPKT_WINDOW bit in our control byte.
*/

oobdata[0] &=~TIOCPKT_WINDOW;
bcopy(cp+4,(char *)&w,sizeof (w)); /*copy into structure*/
```

```
/*and change to host byte order*/
w.ws_row = ntohs (w.ws_row);
w.ws_col = ntohs (w.ws_col);
w.ws_xpixel = ntohs (w.ws_ypixel);
ioctl (pty,TIOCSWINSZ,&w);   /* set the new window size */
return (4+sizeof(w));
}
/*
*slogin server protocol machine.
*
*The only condition for which we can return to the caller is if we get
*an error or EOF on the network connection.
*/
protocol (socketfd, masterfd,sockfd)
int socketfd;  /* network  connection to client for remote login*/
int masterfd;  /*master pseudo terminal*/
int socket;     /*secondry connection to client for file transfer*/
{
        char mptyibuf [1024],sockibuf [1024],*mptybptr,*sockbptr;
register int mptycc,sockcc;
int  cc;
char  cntlbyte;
mptycc=0;       /*count of #bytes in buffer  */
sockcc=0;
/*
* we must ignore SIGTTOU,otherwise we 'll stop when we try and set the slave pty's
*  window size (our controlling ttyis the master pty.
*/
signal (SIGTTOU,SIG_IGN);
/*
 *  send the TIOCPKT_WINDOW control byte to the client( as an 00B data byte )telling it
that we 'll
 * accept window size changes .
 */
send (socketfd,oobdata,1,MSG_00B);
 /*
  *in this loop
  * network input -->sockibuf[  ]
  *                        sokicibuf[  ] --> master  pty (input from client)
  *master pty input  -->mptyibuff[  ]
                           mptyibuff[  ] -->network (output for client)
  *secondry socket input  --> {then file transfer occurs).
  */

  for (;;)
  {
        fd_set ibits,obits,ebits;
FD_ZERO(&ibits);
FD_ZERO (&oBITS );
if ( socked) FD_SET ( masteredfd, orbits);
if  ( mptycc   > =  0)
{
        if  ( mptycc ) FD_SET ( socketfd , & orbits);
        else  FD_SET ( mastered , &iorbits );
```

```
}
FD_SET( socked , &ibits );
FD_ZERO ( &ebits );
FD_SET ( masterefd , &orbit);
if ( select ( 32, & ibits,
FD_ISSET  (socked, &orbits )::
FD_ISSET  ( sockedfd, &orbit )?
&orbit  : (FD_SET   *) NULL ,&ebits,
(struct  timeval * )  0)< 0)
{
if ( errno == ENTER ) continue ;
fatalperror (sockedfd, "{ select  " );
}

if (((! FD_ISSET( socketfd , &orbits )) &&
if (((! FD_ISSET(masterfd , &ibits))&&
if (((! FD_ISSET(sockfd, &ibits ))&&
if (((! FD_ISSET(masterfd , &ibits )))&&
if (((! FD_ISSET(sockfd, &obits )))&&
if (((! FD_ISSET(masterdfd, &ebits)))
{

        /* shouldn't happen */
        sleep ( 5 );
        continue;
}
if ( FD_ISSET ( masterfd, &ebits ))
{
/*
 *There  is an  exceptional condition  on the  master pty . In the pty  packet  mode,
 * this  means  ther is
 * asingle  TIOCPKT_xxxcontrol  byte  to the client  as oob data .
 */

cc  = read (masterfd, &cntlbyte, 1);
If  ( cc==1 &&  pkcontrol (cntlbyte ))
{

cntlbyte & TIOCPKT_FLUSHWRITE)
{
/*
* if  the pty slave  flushed  its  output
*/



* queue , then  we want  to throw away  any thing  we have  in our  buffer  to send  to the
client .  */

mptycc=0;
FD  CLR (masterdfd. &ibits);


                    }
```

```
        }

    }



    if ( FD_ISSET ( sockfd,&ibits ))
    {
    /*
     * There is input ready on the socket from the client .
    */


    sockcc=read (socketfd , sockedibuf, sizeof (sockibuf));
    if ( sockcc < 0&& errno ==EWOULDBLOCK )
                    sockcc=0;
    else
    {
            regester char *ptr ;
    int left , n;

    if ( sockcc <=0; ) break ;
    sockbptr =sockibuf+ sockcc-1;

    ptr++ )
    {
            if ( ptr [0]==magic[0] &&
            ptr[1] ==magic[1])
    {
    /* We have an in-band control message . processit . after we have processed it have to
move all the
    *remaning data in the buffer left , and check for any more in -
    *-band control messages
    */

    left =sockcc-(ptr -sockibuf);
    n=control (mastered, fd, ptr , left );
    if (n)
    {

    left -= n;
    if (left > 0)
    bcopy (ptr+n,
                    ptr,
                    left);
    sockcc -= n;
    goto top ;
                    }
            }
    }
```

```
FD_SET ( masterfd, &orbits ); /* try  write */
                    }
          }

if  (( FD_ISSET ( masterd , & orbits  ))  &&  sockcc   > 0)
{



/*
* The master  pty  is ready  to accept data and there data  from the  socked  to write  to the
mpty.    */

          cc  = write ( masterd fd, sockbptr , sockcc);
if  (cc  > 0 )
{
          sockcc  -= cc;
          sockbptr  +=cc;
          }
}
if   (FD_ISSET (masterfd, &orbits  ))
{

/*
*There  is input  from the master pty. Read  it into  the bigining  of our mptyibuf.buffer.
*/

mptycc  = read (masterd, mptyibuf ,
                    size of ( mptyibuf ));
mptybpt =mptyibuf ;
if ( mpytycc <0  &&  errno ==EWOULBLOCK )
mptycc =0;
else  if  ( mptycc  < =0 ) break ;
else  if ( mptyibuf [0]==0)
{
/*
* if  frist byte is 0, then real  data
*/
 mptybptr++;
mptycc--;
FD_SET( sockefd ,&obits );
}
else
{

/*
 * It's  possible  for the  master  pty to generate  a control  byte  for us , between  the  select
above    and the
 *  read taat we just did .
  */

if   ( pk control ( mptyibuf [0]))
{
mpty ibuf [0]  ,
```

```
1,
MGS_00B);
}
/* else it has to be one of the packet mode control bytes that we' er not intrested */
 * mptycc=0;
/* there can't be any data after the control byte */

                }

        }
If (( FD_ISSET (socketfd,&obits )) && mptycc >0 )
{
/*
* The socket is ready for more output and we have data from the master pty
* to send to the client.
*/


        cc=write (socketfd, mptybptr,mptycc);
if (cc < 0 && errno==EWOULDBLOCK )
{
        sleep (5);
        continue ;
}
if (cc > 0)
{

        mptycc -=cc;
        mptybptr += cc;
        }
}

if ( FD_ISSET 9sockfd, &ibits ))
{
/*
* Recived arequest on the secondary connection for file transfer . Get pwd
* from the login shell and prefix it to the remote file name ( if remote file
* transfer is relate to one ) for a child process and process the file transfer request .
* An exit status of 4 from child process is got when an error has occured and
* the program terminates , other wise continue.
*/

int chipd;
int status ;
signal (SIGCLD, SIG_DLF);
if (( chpid =fork ( ) < 0)
        fatalperror (0,"");
        if (chipd == 0)
{

char buff [MAXFILENAME];
int count , i;
char mesg [20];
```

```
/*
 * set  the uid  and gid  of the  process to the remote  user  so that the process
 *canopen /read writea file
 * only  when  it has  premission to do .
 */


setgid (getpwname(servername ) -> pw_gid );
setuid  (getpwnam (server name)-> pw_uid );

while  ( write (masterfd
"pwd\n"
sizeof  ( " pwd\n")-1)  !=
          size of("pwd\n")-1));
while  ((count =read (masterfd,
                    buff,
          sizeof(buff))) < =1);
                    }
          }
}

for  (i=0;
  i<count && buff [I] ! +'/' );
  i++) ;
  If (i==count )
send _ERROR (ERR_UNDEF,
"system failuer");

else
dir_name =buff+i;

fsm_lop( 0);
exit ( 0) ;
}
status =1;
wait(&status);
if (WEXITSTATUS (status ) ==4 )return;
signal (SIGCLD,&cleanup );
                    }
          }
}
 void clean up ( )

 {
 char * p;
 /*
 * Remove  the /etc/utmp entery by calling  the logout  ( ) fuction . then  add the
 * terminating entry to /usr/adm/wtmp/ file.
 */

 p=line+5;

 if ( logout(p))
    logwtmp(p," "," " );
```

```
chmod( line,0666);      /* change  mode of  slave to rw-rw-rw-  */
chown (line, 0666 );    /* change ower=root*/
*p='p';
chmod(line,0666); /* change  mode of  master to rw-rw-rw-  */
chown(line,0,0 );    /* change ower to root */
  shut down(0,2);  /* close both directio of socket
  shutdown (sockfd2,2);  /*  close both directio of socket
exit (1);

}
/*
*send an error message back to slogin cliient . the first byte must be a binary 1,
* followed by the ASCII
error message  followed by  areturn /newline.
}

/*
* Same  as fatal.
*/
fatalperror (fd,msg)
int fd;
char *msg;
{
fatal (fd,msg );
}
int do_rlogin (host)
char *host;
{
/*
*read the three strings that rcmd1( )wrote to the socket
*/
getstr (cliuname,size of (cliunane),"remuser two long ");
getstr ,(servunane,size of(servunane),"locuser too  long ");
getstr (tern+envsize
sizeof(tern )--ENVSIZE,
"terminal type too long");
if (getuid( ))
return--1;
/*return (ruserok (host,SUPERUSER (pwd),cliuname,servuname));
*/
return 0;
{
/*
*read  a stringfrom the socket
*/
getstr (buf,cnt,errmsg )
char *buf ;
int cnt ;
char  *errmsg ;
}
        char  c;
        do
 {
                if (read (0,&c,1)!=1)
```

```
            exit(1);
        if (--cmt< 0)
                    fatal (1,errmsg);
            *buf++ = c;
            }while (c!=0);
}
extern char **environ;

char *speeds[ ]={

"0","50","75","110","134","150","200","300","600","1200","1800","2400","4800
","9600","19200","38400",}
#define NSPEEDS sizeof( (speeds )/sizeof( speeds [0] ))
/*
*Setup the slave pseudo terminal device.

* We take the terminal that was sent over by slogin client,along with


* speed and and set the speed of slave pty accordlingly.
*/

setup_term (fd)
int fd;

{
            regester char *cp,**cpp;
            struct sgttyb sgttyb;
            char * speed;
            ioctl (fd, TIOCGETP, & sgttyb ); /* fetch modes for slave pty */
            if (( cp=struct (speed ,'/' )) ! =NULL)
{

            * cp++ = '\0' ;
            speed =cp;
            if (( cp=stchr (speeds; cpp< [NSPEEDS] ;cpp++ )
        {
        if (strcmp (*cpp,speed)==0)
                    {
                    sgttyb.sg _ispeed=sgttyb.sg_ospeed ;
                    sgttyb.sg_ospeed=cpp-speeds;
                    break ;
                    }
        }
        }
sgttyb.sg_flags=ECHO: CRMOD:ANYP:XTABS;
            /* echo on */
            /* map CR into LF ; output LF as CR, LF */
            /* accept any parity ,send none */
            /* replace tabs by space on output */
ioctl (fd, TIOCSETP, &sgttyb);
/*
*Initialise the enviroment that we will ask /bin/login to mantain . /bin/login will then
* append its variables
```

```
* ( HOME , SHELLS, USER,PATH, ...)  to this.
*/

env[0]=term;
env[1]=(char  * )0;
environ=env ;
}

/*
*check whether  the specified  host  is in  our local domain,  as determined  by the  part
* of the name following  the first period, in its  name  and in ours . If either  name  is
* unqulified (contains no              period ) assume  that the  host  is local , as it will  be
interupted as such .
*/
 int local _domain (host ) /* return 1 if local  domain , else return 0*/
char  * host;
{
register  char *ptrl, *ptr2;
char  local  host [MAXHOSTNAMELEN];
if (( ptrl=strchr (host, '.'))==NULL))
          return  1;     /* no period  inthe remote   host name  */
gethostname (localhost, size of(localhost));
if  ((ptr2=struchr(localhost ,'.') ) ==NULL)
return  1; /* no period  in local host name */


/*
 * Both host names  contain a period . now compare  both names  starting  with the
 *first period  in each name if  equla , then the  remote domain  equals  the local domain ,
 *return 1;
 */

if  ( strcasecmp (ptr, ptr2 ==0)  /* case  insensitive  compare*/

                    return 1;.
                   return 0 ;

}
```

```
*****************************************************************
                         DEFS.H
*****************************************************************

#include <studio.h>
#include < sys/types.h>
#include <set.jmp.h.

#define MAXBUFF 2048          /* Transmit and recive buffer length
#define MAXDATA 512           /* Max size of data per packet to sen
                             /* 512 is specified by RFC */
#define MAXFILENAME 128     /* Max file name length */
#define MAXHOSTNAME 128     /* Max host name length */
#define MAXLINE 512           /* Max command line length */
#define MAXTOKEN 128          /* Max token length */
/*
*Externels
*/
extern char command[ ];              /* The command being processed
extern char host name [ ];           /* name of host system */
extern jmp_buf jmp_mainloop;        / * To return to main command l
extern int lastsend ;               /* # bytes of data in the last data
extern FILE *localfp ;              /* fp of local file to read or write
extern int modetype ;               /* see MODE_XXX values */
extern int next blknum;             / * next block # to send / recive */
extern int check ;
extern long totnbytes;              /* for get /put statistics printing *

# define MODE_ASCII 0               /* values for modetype */
                                   /* ascii == netascii */
# define MODE_BINARY 1              /* binary == octet */
/*
*One recive buffer and one transimit buffer .
*/
extern char recvbuff [ ] ;
extern char sendbuff [ ] ;

extern int sendlen ;                /*#bytes in send buff [ ] */
/* Define the tftp opcodes . */
# define OP_PQR 1                   /* Read request */
#define OP_WRQ2                     /* Write request */
# define OP_DATA3                   /* Data */
# define OP_ACK 4                   /* acknowledgment */
#define OP_ERROR 5                  /* Error, see error codes below also

#define OP_MIN1                     /*Minium opcode value */
#define OP_MAX 5                    /* Maximum opcode value */
extern int op_sent ;                /* Last opcode sent */
extern int op_recv;                 /* Last opcode recived */
/*
* Define the error codes . These are transmitted in an error packet (
*with an optional netascii error message describing the error .
*/
```

```
#define ERR_UNDEF 0                /* not defined , see error message */
#define ERR_NOFILE 1                       /* File not found */
#define ERR_ACCESS 2                       /* Access violation */
#define ERR_ NOSPACE 3             /* Disk full or allocation exceeded */
#define ERR_BADOP 4                /* Illigal TFTP opration */
#define ERR_BADID 5                /* unknown TID ( port # ) */
#define ERR_FILE 6                 /* File already exists */
#define ERR_NOUSER 7               /* NO such user */
/*
* Define macros to load and store 2_byte integers , since these are used in the TFTP
headers for *opcodes , block numbers and error numbers . these maros handle the
comservation between host *format and network byte odering .
*/
#ifdef lint                        /* hush up lint */
#under ldshort
# undef stshort
short ldshort ( ) ;
#endif                             /* lint */
/*
* Datatype of fuction that don't return an int .
*/
char *get token ( ) ;
FILE *file _open ( ) ;
```

****

FTPCLI.C

****

```
# include " defs.h"
#include <signal.h >
extern  int sockfd2;
ftploop (fp)
FILE   *fp;

{
/*
*Read  acommand  and execute  it and return  to client .
*/
if  ( getline (fp))
{
if (gettoken (command ) != NULL )
                docmd (command );
                }
 }
```

****

CMD.H

****

```
/*
* Header file for user command rpocessing functions.
*/
# include " defs.h"

extern char temptoken [ ] ;        /* temporary token */
 typedef struct Cmds {
char  * cmd _name ;      /*actual command string */
 int (* cmd _ func ) ( ) ;   /* pointer to fuction */
}
Cmds ;
extern Cmds commads[ ] ;
extern int ncmds ;         /* number of elements in array */
```

****

CMDSUBR.C

****

```
# include " cmd,, h"

/* all the following functions are in cmd.c */
int cmd_ascii ( ), cmd_binary( );
int cmd_get ( ) ,cmd _help ( ), cmd_put ( ), cmd_status ( );
extern int sockfd2;

        Cmds commands [ ] = {
                        /* keep in an alphabetical order for binary search */
                "?",            cmd_help ,
                "ascii ", cmd_binary,
                "binary ", cmd_get,
                "get ", cmd_get ,
                "help", cmd_help,
                "mode", cmd_mode,
                "put ", cmd _put ,
                "status ", cmd_status,
                };
#define NCMDS (size of (commands ) / size of (Cmds))

int ncmds = NCMDS;
static char line [MAXLINE] = { 0 };
static char   *lineptr = NULL;
int check;

/*
*Fetch the command line .
*Return 1 if OK, else 0 on error or end of life.
*/
int getline (fp);
FILE *fp;
{
        check =1;
        if (fgets (line ,MAXLINE, fp ) == NULL )
                return ( 0 ) ; /* error or end -of-file */
                linerptr =line;
                return ( 1 );

/*
* Fetch the next token from the input stream .
* we use the line that was set up in the most previous call to get line ( )
* Return a pointer to the token or NULL if no more exit
*/

char * get token (token )
char token [ ] ;
{
register int c;
register char * tokenptr;

while (( c= * lineptr++ ) == '' :: c== ' \t' )
```

```
if  ( c== '\0 : :  c== '\t' ); /* skip  leading  white spaces */
if ( c==  '\0'  :: c== '\n')
return  ( NULL ) ;      /*  no token  */
* tokenptr++  =c;
 While  (( c=  * lineptr ++ )  != `  ` && c! ='\t `  &&c!= `\0')
                    * tokenptr++  =c;
*tokenptr  =  0;             /* null terminate  token */
return ( token );

}
/*
*verify that their are no more token left on the command line.
*/
checkend (  )
{
          if  (gettoken (temptoken) ! =null )
          {
                err_cmd ("trailing garbage");
                 return;
           }
}
docmd (cmdptr )
char *cmdptr;
{
         register int I;
/*
 *get the no of command by performing a binary search on the command array.
 * If not found, then it must be a local command and execute the local command
 * through  system function.
 */
if (( I =binary (cmdptr,ncmds ))<0 )
{
       system (line);
        return ;
}
/*
*call the appropriate function. If all goes well ,that function will return ,otherwise an error
has      occured .In     *this case the error will be displayed and program exits.
*/
(*commands [ I] .cmd_func )( );
/*
 * verify there is no trailing garbage .
*check=0 when the command ( like get ,put) has no third argument .
*In this case there is no need to check for trailing garbage .
 */
if   ( check ) checkend ( );
}
/*
 *perform a binary search of the command table to see if a given token is a command in the
given    table .
 */
binary (word ,n)
char *word;
int n;
```

```
{
register int low ,high ,mid,cond;
low=0;
high =n--1;
while ( low <=high )
{
          mid =(low+high )/2;
         if (( cond=strcmp(word,command [mid ].cmd_name )) <0)
                  high=mid--1;
         else if  (cond >0 )
                  low=mid+1;
         else return (mid ); /*found the command return index in array*/
}
return--1;  /*command not found in command table */
}

/*
 *command error.(like trailing garbage ).
 *print out the command line too,for information.
 */
err_cmd (str )
char *str;
{
         fprintf(stderr," '%s'command error",command );
         if  (strlen (str)>0)
                 fprintf (stderr,": %s",str);
                 fprintf (stderr,"\n");
                 fflush( stderr );
}
```

```
/*****************************************************************************************
                                                         CMD.C
******************************************************************************************
        /*
        * command processing functions.
         */
        extern int check; /*to check for two or one argument to put ,get */ extern int sockfd2;/*
socket for          secondary  connection*/
        /*
         *ascii
         *equivalent to "mode ascii"
         */

        cmd_ascii ( )
        {
           mode type=MODE_ASCII;
           write (1,"\t mode set to ASCII\n",sizeof ("\t mode set to BINARY\n")--1);
        }
        /*
         *binary
         *equivalent to "mode binary"
         */

        cmd_binary ( )
        {
        modetype=MODE_BINARY;
             write (1,"\t mode set to BINARY\n",sizeof("\t mode set to BINARY\n")--1);
        }
        /*
         * get  <remotefilename>  [ <localfilename> ]
         */
        cmd_get ( )
        {
                char remfname [ MAXFILENAME ],locfname [ MAXFILENAME ];
                if (gettoken (remfname)==NULL
                {
                        err_cmd (" the remote filename must be specified");
                        return;
        }
        if (gettoken (locfname )==NULL )
        {
              strcpy (locfname,remfname );
              check=0;
        }
        do_get(remfname,locfname );
        }
        /*
         * help.
         */

         cmd_help ( )
         {
              register int I;
              for  (i=0;i<n cmds;i++)
```

```
        printf (" %s\n",commands [ I ].cmd_name );
    printf ("any local command  (creates a local shell and executes)\n");
}

/*
* mode  ascii
* mode binary
*  Set  the mode  for the file  transfers.
*/
cmd _mode ( )
{
        if ( gettoken  ( temptoken )==NULL )
        {
                err_cmd  (" a mode  type  must  be  be  specified " ) ==0)
                return ;

        }
        else
        {
                if  ( strcmp (temptoken ,  " ascii ")  == 0 )
                {
                        modetype =MODE_ASCII;
                        printf ("\tmode set to ASCII\n");
                }
                else  if ( strcmp (temptoken  , "binary" ) ==0 )
{
        modetype=MODE_BINARY;
printf("\tmode set  to BINARY\n");
{
else
{
        err_cmd(``mode  must be  ascii or binary ``);
        return ;
                }
        }
}

/*
*put  < localfile name> [<remotefilename >].
*/
cmd_put  ( )
{

        char  remfname [MAXFILENAME], locfname [MAXFILENAME];
IF  ( gettoken (locfname ) == NULL )
{
        err_cmd (`` the local file name  must be specified ``);
        return;
}
if  (gettoken (remfname , locname );
check=0 ;
}

do_put (remfname,locfname );
```

```
}

/*
* Show current status.
*/
{
printf (" connected to %s\n" , hostname );
print (``mode =``);
switch(modetype)
{
        case MODE_ASCII :
                                print (" netascii\n"); break;
                case MODE_BINARY :
                                printf("octet (binary)\n" ) ; break ;
                        defult:
                                err_cmd ("unknown modetype");
                        return :
                }
        }
```

****

CMDGETPUT.C

****

```
/*
* File  get /put processing.
*/
# include ``defs. h``
#include  ``time.c``

/*
* execute  a get  command -
* read  aremote  file  and store  on the  local system .
*/

do_get (remfname , locfname )
char  * remfname , * locfname ;
{
        char  mesg[30];
        if (( localfp=file_open (locfname, "w",1))==NULL)
{
spritf (mesg, " can't  fopen  %s for writing " ,locfname );
        perror (mesg);
        return ;
}

totnbytes =0;

t_start ( ) ;                   /* start  timer  for  statictics  */
send _RQ(OP_RRQ,refname ,modetype );
fsm _loop (OP_RRQ) ;
t_stop ( ) ;   /*  stop timer  for  statistics */

file_close (localfp);
                                /*   print  statistics    */
print f(`` Recived % 1b bytes  in  % .if  second  from  %s\n',
        totnbytes, t_getrtime ( ) , hostname ;

}
/*
* Execute  a put  command _send a local  file  to remote system .
*/

do_ put (remfname , locfname)
char * remfname ,*locfname;
                char  mesg [30];
        if (( localfp=file_open (locfname, "r",0)) ==NULL)
{

        sprint ( mesg ," can't fopen  %s for  reading " , locfname );
        perror (msg );
        return ;
}

totnbytes=0;
```

```
t_start ( ) ;                  /* start  time  for  statistic */

lastsend =MAXDATA;
send_RQ9OP_WRQ, refname,modetype);
fms_loop (OP_WRQ);
t_stop ( ) ;   /* stop timer for stastics */
file_close (localfp);
                    /* print statistics  */
printf(" Sent %lb  bytes  in  % .If  seconds  from % s\n",
                    totnbytes , T_getrtime  ( ) , hostname ;
}
```

****

FILE.C

****

```
        /*
        *Routines  to open / close /read/ write a file .
        * For  " binary "  (octet)  transmissions,  we use  the unix  open /read /write  system calls .
        * For "ascii "  (netascii )  transmission , we use  the unix  standrad  i/o routines  fopen
/getc/putc*/

        #include  " defs.h"
        stastic  int  lastcr  =0;        /* 1 if last  chr  was  a carriage _ return */
        static  int  nextchar =0;

        /*
        *Openthe file for  reading  or writing .
        *Return  a file pointer , or NULL on error .
        */

        FILE  * file _open (fname , mode , initblknum )
        char   * fname ;
        char * mode ;  /* for fopen  ( ) - ``r`` or `` w`` */
        int  initblkum ;
        {
        register  FILE  * fp ;
        if  ( strcmp (fname , mode )) ==NULL )
                return (( FILE *) 0);
        /* for  first  data  packet  or first  ACK  */
        next char   =  -1;            /* for file_write ( )*/
        return  ( fp );               /*  for  file _read  ( ) */
        return  ( fp);
        }
        /*
        *Close  the file .
        */


        file _close (fp)FILE  * fp;
        FILE *fp;
        {

                if  (lastcr)
                        {
                        perror ("final character  was  a CR");
                        exit ( 4 ) ;
        }
        if (fp == stdout )
                return;
        else  if  (fclose (fp) ==EOF)
        {
                peror ("fclose error");
         exit ( 4);
        }
```

```
/*
*Read data from the file.
*Here is where we handle we handle any conversion between the file 's mode
 *on the local system and the network mode.
*Return the number of bytes read (between 1 and maxnbytes, inclusive ) or 0on EOF.
 */
int file _read (fp,ptr, maxbytes.mode)
FILE *FP;
register char *ptr;
register int maxbytes; int mode;
{
        regester int c, count;
        if ( mode == MODE_BINARY)
{
        count = read (fileno(fp),ptr, maxnbytes );
        if (count );
        return (count);      /* will be 0 on edf */
}
/*
 * For files that are transferd in netascii ,wemust perform the reserve conversions that_write
 * ( ) dose.
 */

        for (count =0; count < maxnbytes; ;count ++)
{
        if (next char >= 0)
{

        *ptr ++ = nextchar;
        nextchar = -1;
        continue;
}

c=getc(fp);
if (c==(EDF)

{       /* EOF return means eof or error */
if (ferror (fp))
{
perror (" read err from getc on local file ");
exit ( 4 );
}
return count;
}
else if (c=='\r' ;                      /* newline -.> CR, LF */
        nextchr = `/n` ;
}
else if (c==`\r` )
{
        next char = `\0` ; /* CR -> CR, NULL */
}
else nextchr      = -1;
*ptr++ = c;
```

```
        }
return count ;
}
 else
{
        perror (``unknown MODE value``);
        exit  (4);
        }
}

/*
 *Write data to the file .
 * here is where we handle any conversiton between the mode of the file
 *on the network and tjhe local  system ‘s conversations .
 */

file _write ( fp , ptr , nbytes, mode )
FILE *fp;
register char *ptr;
register int nbytes ;
int mode;
{
        regester int c, i ;
        char mesg [30] ;
        if (mode ==MODE _BINARY )
{
                /*
                * For binary mode files , no conversions is requred .
                */

                i=write (fileno ( fp), ptr, nbytes );
                if  ( i !=nbytes )
                {
                sprintf (mesg," write error to local file , i= %d", i );
                perror (mesg);
                exit (4);
                }
        }
                else if ( mode ==MODE_ASCII)
        {
        /*
        * For files that are transfered in netascii, we must perform the following
        *   CR,LF   -> newline ='\n`
        *   CR,NULL  -CR    ='\r`
        *   CR,anything _else -> undefined .
        */
                 for (i=0;i <   nbytes ; i++ )
                c = *ptr++;
                if (lastcr )
                {
                        if (c=='\n`)
                                c'\n`;
                        else if (c=='\0` )
```

```
                        c ='\r';
            else

{
            sprintf (mesg . "CR followed  by 0x%02`` ,
                        c);
                    perror (mesg)  ;
                    exit ( 4 ) ;

            }

            lastcr=0;
            }
            else  if  ( c=='\r' )
            {
                    lastcr=1;
                    continue ;
            }
            if  (putc (c,fp ) ==EOF )
            {
              perror ( `` write  error  from putc  to local file ``);
            exit ( 4 );
                    }
            }
}
            else
                    {
                    perror (``unknown  MODE value ~~) ;
                    exit (4)
            }
}
```

****

FSM.C

****

```
/*
* Finite state machine routines .
*/
# include " defs.h"
# include < signal .h>
# ifdef CLIENT
int recv _ACK ( ) , recv_ DATA ( ) , recv_PQERR ( );
#endif
# ifdef SERVER
int recv_RRQ ( ), recv_WRQ, recv _ACK ( ), recv_DATA ( ) ;
#endif
int fsm_error ( ) fsm_invalid ( ) ;
/*
* Finite state machine table .
*This is just a 2-dimensional array indexed by the last opcode sent and the opcode just
recived theresult is the address of afunction to call to process the recived opcode.
* the 2-d table is
*/
*for client :-
```

for client :-

Packet sent	Packet recived				
	RRQ	WRQ	DATA	ACK	ERROR
RRQ			*		*
WRQ				*	*
DATA				*	
ACK			*		
ERROR					*

for server :-

Packet sent	Packet recived					
	RRQ	WRQ	DATA	ACK	ERROR	
nonein	*	*				
RRQ						
WRQ						
DATA				*		
ACK			*			
ERROR						

```
int ( *fsm _ptr[ OP_MAX+1] [ OP_MAX+1]) ( ) ={
# ifdef  CLIENT
fsm_invalid,
fsm_invalid,
fsm_invalid,
fsm_invalid,
fsm_invalid,
fsm_invalid,
fsm_invalid,
fsm_invalid,
,

fsm_invalid,
fsm_invalid,
recv_ACK,
fsm_invalid,
fsm_invalid,
fsm_invalid,
fsm_invalid,
fsm_invalid,
fsm_invalid,
fsm_invalid,
fsm_invalid,,
fsm_invalid
fsm_invalid,
fsm_invalid,
fsm_invalid,
fsm_invalid,
fsm_invalid,
fsm_invalid,
fsm_invalid,
fsm_invalid,
fsm_invalid,
#enfif /* CLIENT */
#ifdef SERVER
fsm_invalid
fsm_invalid
fsm_invalid
fsm_invalid
fsm_invalid
fsm_invalid
fsm_invalid
fsm_invalid
fsm_invalid
fsm_invalid
fsm_invalid
fsm_invalid
fsm_invalid
fsm_invalid
fsm_invalid
fsm_invalid
fsm_invalid
fsm_invalid ·
fsm_invalid
```

```
fsm_invalid
fsm_invalid
fsm_invalid
fsm_invalid
fsm_invalid
fsm_error
fsm_invalid
fsm_invalid
fsm_invalid
fsm_invalid
fsm_invalid
fsm_error

#end if / * SERVER */
};

/*
*Main loop of the finite state machine .
* For the client , we are called after either an RRQ or a WRQ has been sent
* to the Otherside For the server , we called after either an RRQ or a WRQ
* has been recived from the other side. in this case , the argument will be a 0
*(since nothing has been sent ) but the state table above handles this
*/
int fsm_loop (opcode)
        int opcode
        {

        register int nbytes ;
        char mesg [40] ;

        op_sent =opcode;
        for ( ; ;)
        {
        if (( nbytes =net _recv( recv (recv (recvbuff, MAXBUFF )) <0)
        {
                perror (" net _recvv error ");
                exit (4);
                }

                if (nbytes < 4 )
                {
                        sprint (mesg , " receive length = %d bytes", nbytes );
                perror (mesg);
                exit (4);
        }
                if ( op_ recv < OP_RECV > OP_MIN : : OP_RECV > OP_MAX )
        {
        sprintf (mesg , " invalid opcode recived : %d op _ recv );
                perror ( mesg );
                exit (4 );
        }
/*
 * We call the appropriate fuction , passing the address of the recive buffer and its
length .        *These argument ignore the recived opcode , which we have already processed
```

```
 */
if (( *fsm_ptr[op_recv ])( recvbuff+2, nbytes-2 )  <0)
{
/*
*When  the called  function  returns -1, this loop is done .
*/
return  ( 0 );
   }
  }
}

/*
* Error  packet  recived  and  we were not  expected  it .
*/
int fsm_error (ptr,nbytes )
char  * ptr;
int  nbytes ;
{
          char  mesg [ 40] ;
sprintf ( mesg, "error recived : op_recv =%d``,
          op_sent , op_recv);

          perror (mesg ) ;
          exit  ( 4 );
          }
/*
*Invalid  state  transmission . Some thing is wrong .
*/
int fsm_invalid  ( ptr, nbytes )
{
          charmesg [ 40 ];
sprint (mesg , `` protocol both: op _sent =%d, op_recv =%d;
perror ( mesg );
exit ( 4);
}
```

```
*****************************************************************************************
                                        SENDRECV.C
*****************************************************************************************
        # include " defs.h "
        # include < sys /stat.h>
        # include < ctype .h>

        # ifdef  CLIENT
        /*
        *Send  a read -request  or  a write -request  to the  other system . These  two packets  are
only    *sent by the  client  to the  server . this function  is called  when either  the "get" command
or the  *"put"command  is executed  by the user .
        */
        send _RQ (opcode , fname , mode )
        int  opcode ;                          /* OP_RRQ  or OP_WRQ  */
        char  * fname ;
        int modestr;

         stshort (opcode , send buff);
         strcpy (send buff+2, fnme );
         len =2+strlen ( fname )+1;  /* for null  byte  at end of  fname */
        switch  ( mode )
        {
                case MODE_ASCII : modester =" netascii"; break ;
                case  MODE_BINARY: modest ="octet"; break ;
                defult :
                        err_cmd("unknown  mode ");
                        return ;
        }

                        strcpy (sendbuff+len , modestr );
                        len +=strlen(modestr)+1; /* +1 for null byte at the end  of modestr */
                        send len =len;
                        net _send( sendbuff, sendlen );
                        op_sent=opcode;
        }
        /*
        *Error  packet recived  in responce  to an RRQ or a WRQ. Usally means the file  we are
asking  *for on  the other  system  can't be accused  for some reasons . We need to print  the error
        *message  that 's  returned .
        *called  by finite state machine .
        */

        int  recv _RQERR( ptr, nbtes)
        char  * ptr;                           /* points  just past  recived  opcode */
        int nbytes ;                           /* dose't include received opcodes */

        {
                register  int ecode ;
                ecode= 1dshort (ptr);
                ptr +2;
                nbytes -=2;
                ptr [nbytes ]=0;               /* assured  its nul terminated....*/
```

```
fflush (stdout );
fprint f(stderr, ``Error# %d:  %s\n``, ecode, ptr );
flush (stderr);
                /*terminate  finite state loop */
        return -1;
}
#endif  /* CLIENT*/
/*
* Send an acknowledgement  packet  to the  other syatem . called  by the  recv
_DATA( )        * function  below and  also  called  by recv _WRQ ( ).
*/
send _ACK ( blocknum,)
int  blocknum;
{
        stshort (op_ACK, send buff);
        stshort(blocknum, sendbuff+2);

        sendlen =4;
        net_send (sendbuff,sendlen);

        op_sent=OP_ACK;
        }
         /*
          *Send data to the  other system .
          * the data  must be stored  in tht " sendbuff" by the  caller .
          * Called  by therecv_ack( ) function  below.
          * /
        send _data(blocknum,nbytes)
        int blocknum,sendlen );
        op_sent=OP_DATA;
        }
/*
* Data  packet recived . Send  an acknowledgement
* Called by finite state machine .
* Note that this  function  is called  for both  the client  and the server .
* /

int  recv_DATA (ptr, nbytes )
regester  char *ptr ;               /* points just past recived  opcodes */
regester  int nbytes;               /* dosent include recived  opcodes */

{
register int recvblknum ;
char mesg [40];
recvblnum =ldshot(ptr)
ptr  +=2;
nbytes  -= 2;
if (nbytes  > MAXDATA )
{
        sprintf(mesg, " data packet recived  with  length  = %d bytes ",
        nbytes);
        perror (mesg);
        exit  (4);
        }
```

```
if ( recvblknum == nextblkum )
{
/*
 *The data  packet  is the expected one .
 *Increment  our  expected block  for the next packet.
  */
  nextblknum ++;
  totnobytes += nbytes ;
  if   ( nbytes  > 0)
   {
/*
 *Note  that  the final  data  packet  can have  a data
 * lenth of zero , So  we only  write the data
 * to local  file  if there is  a file
  */

         file _write (local fp, ptr,nbytes , modetype );
         }
#ifdef SERVER
         /*
         If  the length  of the  data  is between  0-511,
         This  is the  last data block . For  the server ,
         here's Where  we have  to close  the file .
         for the client , "get"  command  processing
         will close the file .

         if  (nbytes  < MAXDATA )
         file _close (local fp);

#endif
}
else  if (recvblkum < nextblkum  --1 ))
{
/*
 * We have just recived the data block # N (or  earlier, such as N-1, n-2 ect .) from
the                 *other end  but we were expecting  data block # N+2. But  if  we were
expecting  N+2.                *but if we  were expecting  N+2 it means  we have already  recived
N+1, so  the other*end went  backwards  from N+1  to N (or earlier ) some thing is wrong
 */

    perror (" recvblknum   >   nextblknum");
    exit ( 4 );
    }

    /*
    * The only  case  not  handled  above  is
     * "recvblknum == ( nextblknum -1 )" . this means  the other end
     * never  saw  our  ACK for the  last  data packet  and  retransmitted
     * it . We just ignore  the transmission  and  send  another  ACK .
     *  Acknowledge  the data  packet .
      */
         send _ACK ( recvblkunum  );
      /*
     * If  the length  of the data  is  between  0-511,  we have  just
```

```
* recived  the final  packet , else  there is more  to come .
*/
        return  (( nbytes  == MAXDATA ) ? 0: -1 );
}


/*
* ACK packet  recived . Send some more  more data .
* called  by finite  state  machine . also called  by recv_rrq ( ) to start  the
* retransmission  of afile  to  the client . Note  that this  fuction is
* called  for both the client  and the server .
*/

int recv_ACK (ptr , nbytes )
register  char  * ptr
regester  int  nbtes ;
{
regester  int  recvblknum ;
char  mesg [ 40 ];
recvblknum  = ldshort (ptr ) ;
 if  ( nbytes  !=2 )
{
sprintf  (mesg ,`` ACK packet  recived with lenth  =%d bytes "
        ,nbytes+2);
perror (mesg );
exit(4);
}
if   ( recvblknum == nextblknum )
{
/*
* The recived  acknowledge ment  is fro  the expected  data  that we sent .Fill the transmitted
*buffer  with the next  block  of data  to send  if  there  is no more  data  to send ,  then  then
we      *might  be  finished note  that  we must  send  a finial  data  packet  contaning  0-511 bytes
of data .           *if the  length  of the  last packet  that we  sent  was excately  512  bytes , then  we
must  send a 0    *length data packet.
 /       /*


if ((  nbytes = file _read ( local fp , sendbuff+4  MAXDATA , modetype ))==0)
{
        if ( lastsend < MAXDATA )
                return  -1 ;      /* done */
                /* else  we will send  0 bytes  data  */
        }
        lastsend=nbytes ;
        next blknum ++;  /* incr  for this  new packet  of data  */
        }
        totnbytes  +=nbytes ;
        send _data (nextblknum  , nbytes );
        return  ( 0) ;
        }
        else  if (recvblknum  < ( nextblnum  - 1))
        {
```

```
/*
*We 've just recived the ACK for block # N from the other end , but we were
  expecting the ACK for block # N+2 it means we have already recived the ACK for
  N+1 , so the other end went backwards from N+1 to N . some thing is wrong..


        perror (" recvblknum < ( nextblknum -1 ))
        exit ( 4) ;
        }
        else if (recvblknum > nextblknum )
        {
        /*
        *We'ev just recived the ACK for block # N (or later , such as N+1 , N+2 etc.)
         from the other end , but we were expecting the ACK for block# N-! . but this implies
that    the other end has already recived data block # N-1 from us .
        Some thing wrong
        */
                perror (" recvblknum > nextblknum " );
                exit ( 4);
        }

        else
        {
        * Here we have "revvblkum == (nextlknum -1)" . this means we recived aduplicate ACK
.       This means either :
        ( 1 ) the other side never recived our last data packet ;
        ( 2 ) the other side ACK got delayed some how.
        if we were to retransmit the data packet , we would start the " sorcerer' Apprentice
syndrome     . " we` ll just ignore this duplicate ACK , returning to the FSM loop , which will
initiate another recive .


        return 0;
        }
        /* notreached */

        #ifdef SERVER

        /*
         * RRQ packet recived . Called by the finite state machine
        This ( and reciving a WRQ ) are the only ways the server gets
        started

        int recv_RRQ (ptr , nbytes )
        char * ptr ;
        int nbtes ;
        {
                char ackbuff [2];
                recv_Xrq (op_RRQ, ptr , nbytes );
        /* Set things up so we can just recv -ACK ( ) and pretend we recive an ACK , so it 'll
send    the frist data bock to the client .
        */

        lastsend =MAXDATA;
```

```
stshort ( 0,ackbuff);
recv-ACK (ackbuff ,2);
return  ( 0 );

}
/*
WRQ packet  recived . called  by  finite  state machine . this ( and  recive  an RRQ )
 are the only  ways  the server  gets  started .
*/

nextblknum =1;
send_ACK ( 0 );
return ( 0 );

}
/*
* process an RRQ  or  WRQ that  has  been  recived .
* Called  by the  2 routines above .
*/

int recv_xRQ ( opcode , ptr,nbytes)
int opcode;
regester  char * ptr;
stshort (ecode , send buff+2);
strcpy (sendbuff+4, sendbuff+4)+1;     /* +1 for null at end */
net_send ( send buiff, sendlen );
exit ( 0);
}
/*
* Copy  a starting  and convert  it to lower  case  ijn the processo .
*/

strlccpy (dest ,scr )
regester  char  *dest , *scr;
{
        register  char c;
        while  (( c=  * scr++ )   ! ='\0');
        {
                if   ( isupper  ( c) )
                        c+tolower ( c) ;
                *dest++  =0;
        }
                *dest  =  0;
        }
        #endif /*SERVER*/
```

****

INITVARS.C

****

```
/*
 * Initialize  the external  veriables .
*/

#include  ``defs.h``

char  command [ MAXTOKEN] ={0};
char   hostname [MAXHOSTNAME ]={0};
jmp_buf_mainloop={0};
int lastsend =0;
FILE  *localfp=NULL;
int mode type=MODE_ASCII;
int  nextblknum=0;
int op_sent=0;
int op_recv=0;
char recvbuff[MAXBUFF]={0};
char send buff[MAXBUFF]={0};
int selden=0;
char temptoken[0]={0};
long totnbytes =0;
```

****

NETTCP.C

****

```
#include " rw.c"
extern int sockefd2;
/*
* Close the file transfer net work connection .
*/

net _close ( )
{
        close (sockfd2);
        sockfd2= -1;
}

/*
*send arecord to the other end. Used by the client and the server .
*we prefix each record with its length .
* we encoded the length as a 2-byte integer in network byte order .
*/
net_send (buff,len)
char* buff ;
int len;
{
        register int rc ;
        short templen ;
        templen =htons(len);
        rc=writen =(sockfd2, (char *)& templen , size of short ));
        if ( rc ! =size of (short ))
        {
        perror (`` written error of length prefix``);
        exit ( 4 );
        }
        rc=written (sockfd2, buff,len );
        if ( rc != len)
        {
                perror ( ``written error ``);
                exit (4 );
        }

}

/*
 * Receive arecord from the other end . used by client and server .
*/

int net_recv(buff,maxlen)
char * buff;
int maxlen ;
{
        regester int nbytes ;
        short templetn;
```

```
again 1:
        if (( nbytes =readn (sock fd2, (char *  )& templen , size  of (short ))) <0)
{

        if (errno ==Einter )
        {

        errno=0;    /* assume SIGCLD  */
        goto  again ;
        }
perror (``readn  error  for  length  prefix ``);
exit  ( 4);
}
if  ( nbytes ! = size of length  prefix");
{

        perror (`` error  in readn of length  prefix``);
        exit (4);
}
templen =ntohs  (templen );          /* # bytes  that follow */
if  (templen  >  maxlen  )
{
        perror (``record  length  too large ``);
        exit (4);
}
again2:
        if ((nbytes ==readn (sockfd2, buff,templen ``))  < 0 );
        {
                (errno ==ENTER)
                {
                error ==0;  /* assume  SIGCLD */
                goto  again 2;
                }
                perror (``readn  error ``);
                exit (4);
        }
                return (nbytes) ;  /* return  the actual  length  of the  message */


ntohs  ( templen );
```

```
            if (eerrno==EAGAIN)
                    fprintf(stderr,
                            ``socket: ALL  PORTS inuse \n``);

            else
                    perror ("rcmd: socket ");
            sigsetmask (oldmask );
            return -1;
    }
    /* Set pid  for socket  signals  */
    fcntl (socked1 ,F_SETOWN, getpid ( ));
    /*Fill  in the  socket  address of the  server ,
     and connect  to the server    */
    bzero (( char  * ) & server _addr , size  of (serv_addr));
    serv_addr  .sin _famaly = hp-> _addrtype ;
    bcopy ( hp->h__addr list [0],
            (caddr_t)& serv_addr.sin_addr,
    hp->h_length );
    serv_addr.sin _port=rport;

    if (connect (sockfd 1,
                    (struct sockaddr   * ) &serv_addr,
                    size of  (serv_addr))  >=  0)
                    break ;   /*  o.k., continue onword  */

    close (sockfd1);
    if    (error==EADDRINUSE)
    {

            /* We were  able  tobind the local  address, but could not  connect  to the server .
            decrement  the  starting  port no  for  rresvport ( ) and try again */
            1port ---;
            continue;

    }
            if errno ==ECONNREFUSED  &&timo  <= 16
    {
            /*  the connection  was  refused . the server 's  system  is probably  over load . sleep
for  a      while , then try again .we try this   5 times (total  31 seconds )*/

            sleep (timo );
            timo * =2;   /* increse timer  :1,2, 4,8,16, sec * /
            continue
    }
    if   ( hp-> h_addr_list[1]  ! NULL )
    {
    /* if  there is another
    address  for the host  try it */
    int oerrno =errno;
    fprint f(stderr,
                    `` connect  to address  %s ``,
                    inet_noat(serv_addr.sin_addr,
                    hp-> _length );
    fprintf(st          dderr
```

```
*************************************************************************************
                                   RCMD.H
*************************************************************************************
#include <sys/types-h>
/*
*fordefinitions of FD-xxxmacros include sys /types-h and sys/time-h
*/
#include<sys/time``h>
#include <sys/socket-h>
#include<sys/file-h>
#include<sys/file=h>
#include<netinet/in=h>
#include<arpa/inet=h>
#include<stdio:h>
#include<netdb:h>
#include <errno:h>
extern int errno:
/*Return socket descriptor-sockfd*/
int rcmd 1(ahost:rport „cliuname„servuname„cmd„fd2ptr)
/*Pointer to address of host name */
char **ahost;
/*Port on server to connect to -network byte order*/
unsigned short rport;;
/*Username on client system (i:e :cller s username*/
char *cliuname:;
/*Username touse on server system*/
char *servuname:
/* Command string to execute onthe server */
char * cmd ;
/* ptr to secondary socket descriptor ( if not NULL )* /
int * fd2ptr;
{

int socked 1, timo , 1port ;
long oldmask ;
char c;
struct sockaddr _in serv_addr, serv 2_addr ;
struct hostent *hp ;
fd_ set readfds ;
if (( hp=gethostbyname (*ahost )) ==NULL )
{

        perror (`` at gethostbyname 11);
        return -1;
}

oldmask =(sigblock ( sigmask (SIGURG )));

1 port =IPPORT _RESERVED -1;
timo=1;
for (;;)
{
if ((sockfd1 =rresvport (&1port )) < 0)
{
```

```
                " Trying  %s....\n,
                    inet_noat (serv_addr.sin_addr));
            continue ;
    }
    perror ( hp -> f_name ;  /* none of the above , quit  */
    sigsetmask (oldmask );
    return   -1 ;
    }
    if ( fd2ptr == (int * )0)
    {
    /*
    * caller dose't want asecondary channel . write
    *a byte  of 0 to the socket, to let the server  know this
    */

    write (sockfd1, "  ",1);
    1 port=0;
    }
    else
    {
            /* Create  the secondary  socket and connect  it to the server  also .
            we have  to bind  the secondary  socket to a reserve TCP port also */
            char num [8];
            int socktemp,sockfd2,len;
            1port --;       /* decrement for starting  port  #*/
            if ((socktemp ==reserved ==rresvport (&1port ))  <0)
                    goto bad;
            listen socktemp,1);
            /* Write


    /*caller does t want a secondary channel.write
    *abyte of 0 to the socket,to let the server know
    */
    write (sockfdi,,'',1);
    1port=0;
    }
    else
    {
    /*create the secondary socket and connect it to the server also .We have to bind the
secondary          socket to a reserved TCPport also */char num[8];
    int sock temp,sock f d 2, len;
    1port--;/*decrement for starting port#*/
    if((sock temp=rresvport( 1port))<0)
       goto bad;
    listen(sock temp,1);
    '*write an ASCIIstring with the port number to the server so it knows which port to
connect*/print f (num,%d;,1port);
    if (write (sock fd1,num,str 1en(num+1)=str len(num)+1)
    {
            perror('write;setting up stderr');
    close(sock temp);
    goto bad;
    }
```

```
FD-ZERO (&read fds);
FD-SET(sock fd1,&read fds);
errno=0;
if ((select(32,
             &read fds,
            fd-set*)0
           (fd-set *)0
          (strict to,eva; *)0)<1)
       FD-1SSET{sock temp,&read fds};
}
       if(errno=0)
      perror(select;setting up secondconnection);
else
fprint f(stderr,
      select
     protocol failure in circuit setup.\n);
close(sock temp);
goto bad;
}
/*The server does the connect() to us on the secondary socket */
len=sizeof (ser v2addr);
sock fd2=accept (sock temp,
                 (struct sockaddr *)ser v2-addr,
                  &1en);
close(sock temp);/*done with this descriptor*/
if (sock fd2<0};
{
 perror("accept ") ;
lport =0,
goto   bad ;
}
*fd2ptr    =sockfd2;  /* to return  to caller  */
/*  the server  has  to its  end of  this  connection  to a
reserve port also, or we don't  accept  it */
serv 2_ addr.sin _port =
noths ( unsigned  short  ) serv2_ addr.sin_port );
if (( serv2_addr.sin famaly  ! =af_INET ) : :
(serv2_addr.sin_port > IPPORT _RESERVED )::
(serv2_addr.sin_port <  IPORT_RESERVED/2))
{
          fprint (stderr ,
                   " socket : protocol  failuer  in circuit  setup .\n");
                   goto bad 2;
}
          printf ( "\nescondary  connection  established  \n");
}

write (sockfd1, cliuname ,strlen (cliuname )+1);
write (sockfd1,servername , strlen (servuname ,strlen (servname)+1);
write (sockfd1 ,cmd ,strlen (cmd)+1);
if (read ( sockfd 1, &c,1)  ! = 1) /* read one byte from server */
{

          perror (*ahost );
```

```
        goto bad 2
}
if (c != 0)
{

/* We didn't get back the byte of zero. There was an error detected
* by the server . Read every thing else on the socket up through anew
*line Which is an error message from the server,and copy to the sender
*/

while ( read (sockfd1,&c,1)==1
{
write (2, &c,1);
if (c=='\n')break ;
}

goto bad 2;
}
sigsetmask (oldmask );
return sockfd 1; /* all OK , return socket descriptor */
bad2:
if ( 1port ) close ( * fd2ptr );
/* then fall through */

bad : close (sockfd1 );
sigsetmask (oldmask );
return -1 ;
}
```

****

RW.C

****

```
/*
*Read nby tes from descriptor;
*use in place of read when fd is a stream socket
*/
int readn(fd:ptr:nby tes)
register int fd:
register char *ptr:
registerint nbytes:
{
     int nleft,nread:
     nleft=nbytes;
    while(nleft>0)
{
nread=read(fd,ptr,nleft);
if (nread<0)
return(nread);/*error,return<0*/
else if (nread==0)
break;/*EOF*/
nleft-=nread;
ptr+=nread;
}
return(nbytes-nleft);/*Return >=0*/
}
/*
*write nbtes from descriptor:
*use in place of write ( )when fd is a stream socket
*/
int writen(fd ptr nbytes)
regester int fd;
regester char *ptr ;
regester int nbytes ;
{
        int nleft, nwritten ;
        nleft = nbytes ;
        while ( nleft > 0)
{
        nwritten = write (fd, ptr,nleft);
        if (nwritten < =0 )
        return (nwritten );      /* error */
        nleft -=nwritten ;
        ptr += nwritten ;
        }
        return (nbytes-nleft);
}
/*
* Read aline from a descriptor .Read the one byte at atime , looking for
* the newline . We store the newline in the buffer , then follow it with a
* NULL ( the same as the fgets (3) ).We return the number of character
*up to, but not including , the null (the same as strlen (3) )
*/
```

```
int  readline (fd,ptr,maxlen)


Register char*ptr:
Register int maxlen:
{
   int n,rc;
char c;
for(n=1;n<maxlen; n++)
{
if ((rc=read(fd,&c,1))==1)
{
*ptr++=c;
if(==\n)break;
}
else if(rc==0)
{
if(n==1)return0;/*EOF,no date read */
else break;/*EOF,some data was read */
}
else return-1;/*error*/
else return -1/*error*/}
*ptr=0;
return n;
}
```

****

TIME .C

****

```
# include < studio.h>
#include < sys /types .h>
#include < sys/param.h>          /* Need the definition of hz */
#define TICKS HZ                 ? * usally 60 or 100 */

static long time _start , time _stop ;
static struct tms _start , tms _stop ;
long time ( ) ;
static double seconds :
/*
* start the timer .We don't return any thing to the caller ,
We just store some information fort some timer routine to access.
*/

void t_ start ( )
{
        if (( time _start = time ( &tms _start )) == -1 )
{
        perror (" t_start : times ( ) error " );
        exit ( 4)

}

}
/*
* stop the timer and save the appropriate information .
*/
void t_stop ( )
{
        if ((time _stop =times ( &tms_stop )0 == -1 )
{               perror(" t_stop : times ( ) error " );
        exit ( 4);

}
}
/*
Return the real ( elasped ) time in seconds .
*/
double t_get rtime ( )
{
        seconds =(double ) time _stop - time _start ) /( double )TICKS
return seconds
}
```

```
*****************************************************************************************
                              CLIENT.C
*****************************************************************************************
*

#include  CLIENT
#include  " rcmd .h "
#include  < sys/param .h>
#include  < sys/file.h>
#include  < sys/socket.h>
#include  < sys/time.h>
#include  < sys/recource .h>
#include < sys/types.h>
#include  <sys/wait.h>

#include  < netinet/in.h>

#include  < stdio.h >
#include  <bsd/sgtty.h>
#include  < pwd.h>
#include  <signal.h>
#include <setjmp.h>
#include < netbd.h>
#include  <error.h>


#include  " ftpcli.c"
#include " cmdsubr.c"
#include  "cmd.c"
#include  "initvars"
#include " cmdgetput.c"
#include  " file.c"
#include   "fsm.c"
#include  " senddrcv.c"
#include  "nettcp.c"
extern int erno;

/*
* The server  sends  us a TIOCPKT_WINDOW notification  when  it starts up.
Thevalue  for this (0x80) can't  over lap  the kernel  defined
TIOCPKT_xxx values

#defined  TIOCPKT_WINDOW
#define TIOCPKT_WINDOW 0x80
#endif

char * struchr ( ) ,* strrchr , ( ) * getenv ( ) , * strcat ( ) , * strcpy ( ) ;
struc passwd  * getpwuid ( ) ;

char  * name ;
int sockfd;                    /* socet to server */

char escchar  ='~';
int eight ;
```

```
int ttout;
char * speeds[ ] ={
        "0" " 50" "75" "110" "134" "150" "200" "300" "600" "1200" "1800"
        "2400" "4800" "9600" "19200" "38400"
        };
char term [256] ="network" ;
int dosigwinch=0;
        /* set to 1 if the server support our window -size-change protocol */

#ifndef sigmask
```