NETWORK MANAGEMENT TOOLKIT

PROJECT WORK DONE AT

KUMARAGURU COLLEGE OF TECHNOLOGY

PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENT FOR THE AWARD FOR THE DEGREE OF

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

OF BHARATHIAR UNIVERSITY, COIMBATORE

SUBMITTED BY

V.ANURADHA 9927K0115

N.M.VINOTH 9927K0175

GUIDED BY

MS.V.VANITHA M.E., LECTURER

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

COIMBATORE – 641 006

2002-2003

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

(AFFILIATED TO BHARATHIAR UNIVERSITY,COIMBATORE)

CERTIFICATE

THIS IS TO CERTIFY THAT PROJECT WORK ENTITLED

NETWORK MANAGEMENT TOOLKIT

IS A BONAFIDE RECORD OF WORK DONE BY

V.ANURADHA 9927K0115

N.M.VINOTH 9927K0175

IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD OF BACHELORS DEGREE IN COMPUTER SCIENCE AND ENGINEERING OF BHARATHIAR UNIVERSITY, COIMBATORE

.....
PROFESSOR AND HEAD
(DR.S.THANGASAMY)

.....
PROJECT GUIDE
(V.VANITHA)

SUBMITTED FOR UNIVERSITY EXAMINATION HELD ON 17.03.2003

.....

.....

ogy
orks
you

SSi
It's where your future is.

14 March 2003
Coimbatore - 18

CERTIFICATE

This is to certify that Mr. N.M. VINOTH and Ms. V. ANURADHA Final Year B.E (Computer Science) students of Kumaraguru College of Technology, Coimbatore has successfully completed the project entitled "NETWORK MANAGEMENT TOOL KIT", working to our complete satisfaction.

Their behavior and conduct during the period of their project work in our company was excellent.

For SSi Ltd.,

Project Coordinator

ACKNOWLEDGMENT

We oblige our Principal **Dr. K. K. Padmanabhan, B.Sc.(Engg), M.Tech., Ph.D** for providing the necessary facilities, which enabled us to complete the project.

We wish to make a special acknowledgment and thanks to **Dr. S. Thangaswamy Ph.D.,** Head of the Department of Computer Science and Engineering for the benefit of his wide engineering experience, knowledge of his work and assessment to implement this project.

We would like to thank our guide **Ms. V. Vanitha, M.E.,** Lecturer, Department of Computer Science and Engineering without whose motivation and guidance we would not have been able to embark on a project of this magnitude.

We would also like to thank our class Advisor **Mr. M.N.Gupta, Mr. M. Vinod and Mr. Dhanasekaran** for their immense help and valuable inputs for the successful completion of our project.

We would also like to thank our project guide in SSI **Mr. Bala** for his immense support for successful completion of our project.

DEDICATED TO OUR BELOVED
PARENTS AND WELL-WISHERS

SYNOPSIS

The early 1980s saw tremendous expansion in the area of network deployment. As companies realized the cost benefits and productivity gains created by network technology, they began to add networks and expand existing networks almost as rapidly as new network technologies and products were introduced. By the

Mid-1980s, certain companies were experiencing growing pains from deploying many different (and sometimes incompatible) network technologies.

The problems associated with network expansion affect both day-to-day network operation management and strategic network growth planning. Each new network technology requires its own set of experts. In the early 1980s, the staffing requirements alone for managing large, heterogeneous networks created a crisis for many organizations. An urgent need arose for automated network management (including what is typically called network capacity planning) integrated across diverse environments.

So the living in a world where network management has become the watchword of the day, here is an attempt to implement few of the network management features in a customized environment.

TABLE OF CONTENTS:

CURRENT STATUS OF THE PROBLEM

CURRENT STATUS OF THE PROBLEM:

Enterprise networks are made up of many devices, some of which are critical to network operation while others are valued for the services they provide. With such a large number and variety of devices, configuring and maintaining each one manually would require a huge amount of manpower and time, and would ultimately cost a great deal of money, here comes the need of network management and network management system.

In the present scenario of network management there exists no commercial package binding all monitoring and administrative features in a Linux Network along with traffic analyzer in Ethernet. Also no package exists to capture client screen at each instant and to control the client's applications from the server, all existing packages consist of only one function either r monitoring or administration or traffic analysis. In a simple network if a source has to be installed to the entire client's it has to be installed in the entire client separately which is time consuming and involves lot of effort, no tool exists to install a source application to all client from the server. Taking backup at the server frequently increases network traffic and no alternate solution is provided to reduce this traffic.

INTRODUCTION

INTRODUCTION

To begin with this: "There is nothing new under the sun "This quote is brilliant statement made by William Shakespeare. It is brilliant because, in literature that preceded his own, for thousands of years, the statement had already been made. Therefore, he used a redundancy to articulate redundancy. How does this relate to Network Management? Read on.

The truth is, TCP/IP has been around for a long, long time. For example NetWare had fully functional TCP/IP built into its operating system back in 1991. UNIX had it for far longer. So there is no real problem here. The knowledge is available out there in the void.

The greater majority of network breaches stem from human error. These human errors generally occur from lack of experience. The techniques to protect a Network have not significantly changed over the past few years. Let us discuss certain network managerial concepts before we go into more of the project.

NETWORK MANAGEMENT BASICS

Network management means different things to different people. In some cases, it involves a solitary network consultant monitoring network activity with an outdated protocol analyzer. In other cases, network management involves a distributed database, autopolling of network devices, and high-end workstations generating real-time graphical views of network topology changes and traffic. In general, network management is a service that employs a variety of tools, applications, and devices to assist human network managers in monitoring and maintaining networks.

The network management consists of five conceptual areas:

- Performance Management
  - The goal of performance management is to measure and make available various aspects of network performance so that internetwork performance can be maintained at an acceptable level.
- Configuration Management
  - The goal of configuration management is to monitor network and system configuration information so that the effects on network operation of various versions of hardware and software elements can be tracked and managed.
- Accounting Management
  - The goal of accounting management is to measure network utilization parameters so that individual or group uses on the network can be regulated appropriately.
- Fault Management
  - The goal of fault management is to detect, log, notify users of, and (to the extent possible) automatically fix network problems to keep the network running effectively.
- Security Management
  - The goal of security management is to control access to network resources according to local guidelines so that the network cannot be sabotaged (intentionally or unintentionally) and sensitive information cannot be accessed by those without appropriate authorization.

TCP/IP BASICS :

TCP/IP is made up of two acronyms, TCP, for Transmission Control Protocol, and IP, for Internet Protocol. TCP handles packet flow between systems and IP handles the routing of packets. All modern networks are now designed using a layered approach. Each layer presents a predefined interface to the layer above it. By doing so, a modular

The ISO/OSI protocol with seven layers is the usual reference model. Since TCP/IP was designed before the ISO model was developed it has four layers; however the differences between the two are mostly minor. Below, is a comparison of the TCP/IP and OSI protocol stacks:

OSI PROTOCOL STACK

7.	Application	--	End user services such as email.
6.	Presentation	--	Data problems and data compression
5.	Session	--	Authentication and authorization
4.	Transport	--	Guarantee end-to-end delivery of packets
3.	Network	--	Packet routing
2.	Data Link	--	Transmit and receive packets
1.	Physical	--	The cable or physical connection itself.

TCP/IP PROTOCOL STACK.

5.	Application	--	Authentication, compression, and end user services.
4.	Transport	--	Handles the flow of data between systems and provides access to the network for applications via the (BSD socket library)
3.	Network	--	Packet routing
2.	Link	--	Kernel OS/device driver interface to the network interface on the computer.

BELOW ARE THE MAJOR DIFFERENCE BETWEEN THE OSI AND TCP/IP:

The application layer in TCP/IP handles the responsibilities of layers 5, 6, and 7 in the OSI model.

The transport layer in TCP/IP does not always guarantee reliable delivery of packets as the transport layer in the OSI model does. TCP/IP offers an option called UDP that does not guarantee reliable packet delivery.

SOFTWARE COMPONENTS OF TCP/IP

APPLICATION LAYER:

Some of the applications we will cover are SMTP (mail), Telnet, FTP, Rlogin, NFS, NIS, and LPD

Transport Layer:

The transport uses two protocols, UDP and TCP. UDP which stands for User Datagram Protocol does not guarantee packet delivery and applications which use this must provide their own means of verifying delivery. TCP does guarantee delivery of packets to the applications which use it.

Network Layer:

The network layer is concerned with packet routing and used low level protocols such as ICMP, IP, and IGMP. In addition, routing protocols such as RIP, OSPF, and EGP.

Link Layer:

The link layer is concerned with the actual transmittal of packets as well as IP to ethernet address translation. This layer is concerned with Arp, the device driver, and Rarp.

SOCKETS:

Sockets are a mechanism for exchanging data between processes. These processes can either be on the same machine, or on different machines connected via a network. Once a socket connection is established, data can be sent in both directions until one of

ABOUT THE PROJECT:

This project consists of three basic modules each having a specific task to be carried out. The three modules are

1.Client Monitoring
2. Network Administration
3.Traffic Analysis

In the basic Network Management Software there are five management stages namely configuration management, accounting management, fault management, performance management and security management. All these five management stages are split up among this Network Management Toolkit's modules.

CLIENT MONITORING:

In case of the first module namely Client Monitoring, there are several sub-phases dealt with. This phase is developed so as to perform the monitoring process over the client so that the status of the clients and other detailed information of the client are available at the server at any instant for the administrator.

The first sub-phase is the basic messaging facility between the client and the server. The messaging facility is provided so that the server can readily communicate to clear their doubts or to perform any function.

The second sub-phase of this monitoring module is the client screen capturing process by the server. The screen capturing facility is provided so that any job performed at the client can be viewed at the server as a picture file rather than only text information. This process is especially useful to find out if the client is misusing the facilities provided.

The third sub-phase is the file transfer process. In this phase the client can request for any file that is required by its local user from the server and after obtaining it can use

The fourth sub-phase of this module is the availability of each of the clients hardware information, process information, and all other services information at any instant at the server. This is provided for the administrator to keep track of the services and process information of all clients.

The last sub-phase of this module is the chatting facility available between the client and the server. This is provided so that the user and the administrator can exchange their views about any topic and also find solutions to their queries.

NETWORK ADMINISTRATION:

The next module is the Network Administration module for handling the administrative functions of the network and is sub-divided into several functions like

-CD installation
-Client Backup
-Services inactivation

The first sub-phase, CD installation, is done so as to share software that is present in the server to all clients by installing it in each of the clients local hard disk. In this phase using a single source rpm file that is present in the server, the clients which require that rpm would request for installation and the server would do it by transferring the rpm file to those clients and installing it.

The next sub-phase of this administration module is the process of backup of the user's file information in the client itself at logout and also in the server at the same instant. This is done so as to have a backup copy of all users information files so that there can be a recovery of users information during system crash and also backup at client is available at each users logout so that it avoids traffic at the communication lines to the server due to backup being transferred to the server each time.

The final sub-phase of this second module is killing of unwanted processes

applications is done by the server so that if the user at any particular client is misusing the facilities available.

TRAFFIC ANALYSIS:

The last module is Traffic Analysis module which is done so as to find out which client in the network is producing maximum traffic. This is done by measuring the number of packets transferred from the client through its Ethernet card to the server.

LITERATURE SURVEY

LITERATURE SURVEY

Network management means different things to different people. In some cases, it involves a solitary network consultant monitoring network activity with an outdated protocol analyzer. In other cases, network management involves a distributed database, autopolling of network devices, and high-end workstations generating real-time graphical views of network topology changes and traffic. In general, network management is a service that employs a variety of tools, applications, and devices to assist human network managers in monitoring and maintaining networks.

A HISTORICAL PERSPECTIVE

The early 1980s saw tremendous expansion in the area of network deployment. As companies realized the cost benefits and productivity gains created by network technology, they began to add networks and expand existing networks almost as rapidly as new network technologies and products were introduced. By the Mid-1980s, certain companies were experiencing growing pains from deploying many different (and sometimes incompatible) network technologies. The problems associated with network expansion affect both day-to-day network operation management and strategic network growth planning. Each new network technology requires its own set of experts. In the early 1980s, the staffing requirements alone for managing large, heterogeneous networks created a crisis for many organizations. An urgent need arose for automated network management (including what is typically called network capacity planning) integrated across diverse environments.

NETWORK MANAGEMENT ARCHITECTURE

Most network management architectures use the same basic structure and set of relationships. End stations (managed devices), such as computer systems and other network devices, run software that enables them to send alerts when they recognize problems (for example, when one or more user-determined thresholds are exceeded). Upon receiving these alerts, management entities are programmed to react by executing

Management entities also can poll end stations to check the values of certain variables. Polling can be automatic or user-initiated, but agents in the managed devices respond to all polls. Agents are software modules that first compile information about the managed devices in which they reside, then store this information in a management database, and finally provide it (proactively or reactively) to management entities within network management systems (NMSs) via a network management protocol. Well-known network management protocols include the Simple Network Management Protocol (SNMP) and Common Management Information Protocol (CMIP). Management proxies are entities that provide management information on behalf of other entities. Figure depicts a typical network management architecture.

ISO NETWORK MANAGEMENT MODEL

The ISO has contributed a great deal to network standardization. Its network management model is the primary means for understanding the major functions of network management systems. This model consists of five conceptual areas, as discussed in the next sections.

PERFORMANCE MANAGEMENT

The goal of performance management is to measure and make available various aspects of network performance so that internetwork performance can be maintained at an acceptable level. Examples of performance variables that might be provided include network throughput, user response times, and line utilization.

Performance management involves three main steps. First, performance data is gathered on variables of interest to network administrators. Second, the data is analyzed to determine normal (baseline) levels. Finally, appropriate performance thresholds are determined for each important variable so that exceeding these thresholds indicates a network problem worthy of attention.

Management entities continually monitor performance variables. When a performance threshold is exceeded, an alert is generated and sent to the network management system.

Each of the steps just described is part of the process to set up a reactive system. When performance becomes unacceptable because of an exceeded user-defined threshold, the system reacts by sending a message. Performance management also permits proactive methods: For example, network simulation can be used to project how network growth will affect performance metrics. Such simulation can alert administrators to impending problems so that counteractive measures can be taken.

CONFIGURATION MANAGEMENT

The goal of configuration management is to monitor network and system configuration information so that the effects on network operation of various versions of hardware and software elements can be tracked and managed. Each network device has a variety of version information associated with it.

Configuration management subsystems store this information in a database for easy access. When a problem occurs, this database can be searched for clues that may help solve the problem.

ACCOUNTING MANAGEMENT

The goal of accounting management is to measure network utilization parameters so that individual or group uses on the network can be regulated appropriately. Such regulation minimizes network problems (because network resources can be apportioned based on resource capacities) and maximizes the fairness of network access across all users.

As with performance management, the first step toward appropriate accounting management is to measure utilization of all important network resources. Analysis of the results provides insight into current usage patterns, and usage quotas can be set at this

From this point, ongoing measurement of resource use can yield billing information as well as information used to assess continued fair and optimal resource utilization.

FAULT MANAGEMENT

The goal of fault management is to detect, log, notify users of, and (to the extent possible) automatically fix network problems to keep the network running effectively. Because faults can cause downtime or unacceptable network degradation, fault management is perhaps the most widely implemented of the ISO network management elements.

Fault management involves first determining symptoms and isolating the problem. Then the problem is fixed and the solution is tested on all-important subsystems. Finally, the detection and resolution of the problem is recorded.

SECURITY MANAGEMENT

The goal of security management is to control access to network resources according to local guidelines so that the network cannot be sabotaged (intentionally or unintentionally) and sensitive information cannot be accessed by those without appropriate authorization. A security management subsystem, for example, can monitor users logging on to a network resource and can refuse access to those who enter inappropriate access codes.

Security management subsystems work by partitioning network resources into authorized and unauthorized areas. For some users, access to any network resource is inappropriate, mostly because such users are usually company outsiders. For other (internal) network users, access to information originating from a particular department is inappropriate. Access to Human Resource files, for example, is inappropriate for most users outside the Human Resources department.

Security management subsystems perform several functions. They identify sensitive network resources (including systems, files, and other entities) and determine mappings between sensitive network resources and user sets. They also monitor access

PROPOSED LINE OF ATTACK

PROPOSED LINE OF ATTACK:

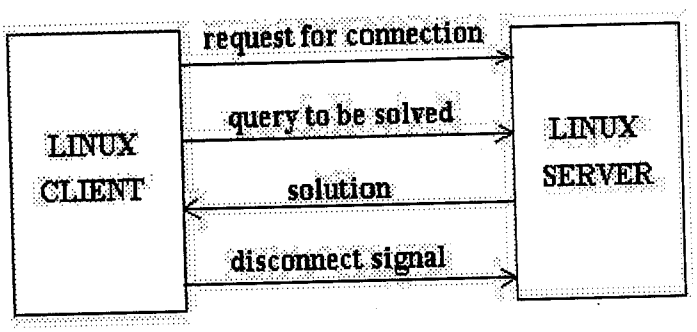
CLIENT MONITORING:

MESSAGING AND CHATTING:

The input to this phase is the request from the client to contact the server for resolving its queries.

The processing is done using a C/C++ program at the client side and the server side separately and establishing socket connectivity between server and client.

The output is the connection establishment and the messages being transferred between the client and the server.



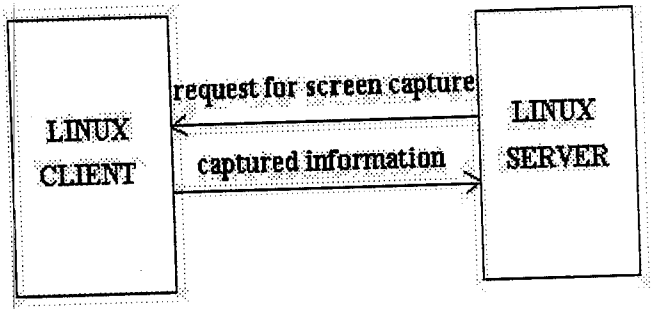
SCREEN CAPTURING:

The input to this phase is the client's ip address whose screen is to be captured.

The processing done is by establishing a socket connectivity using a C/C++ program in the client and server separately and saving it in their .profile files so

network's port frequency matches then that specific clients screen is captured and displayed at the server.

The output of this phase is the jpeg file / picture file of the client screen information of the client displayed at the server.

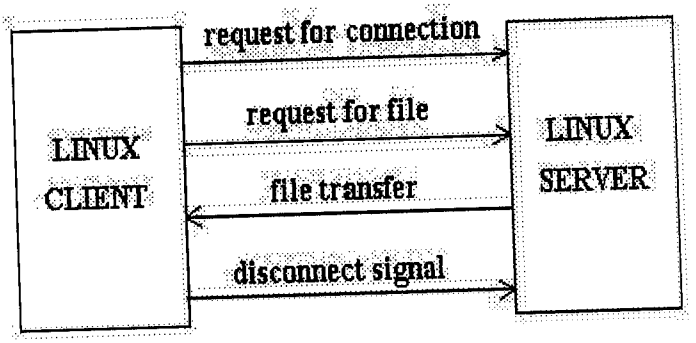


FILE TRANSFER:

The input to this phase is the request of file to be transferred from the server to the client.

The processing to be done is by establishing the socket connectivity using a C/C++ program both at the client and the server separately and then the requested file is transferred from the server to the client with the same port frequency.

The output is that the file is transferred from the server to the requesting client and is available at the client's local memory for its purpose.

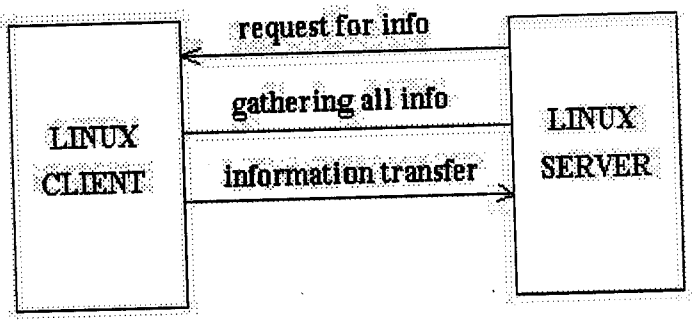


CLIENT INFORMATION:

The input to this phase the ip address of the client whose hardware, software, process, network, services and memory information are to be sent to server.

The processing to be done is by establishing socket connectivity between client and server by using a C/C++ program at the server and client side separately and by obtaining the information at the client and then transferring to the server.

The output of this phase is that all the clients(in the network) complete information is available at the server.



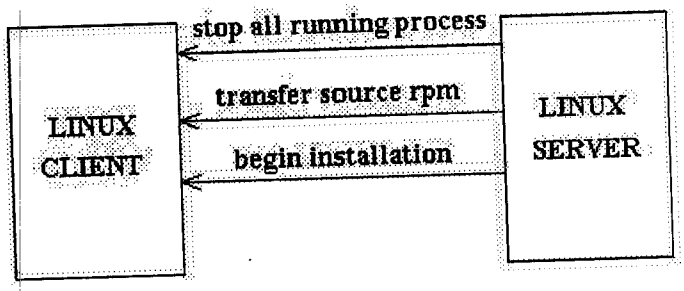
NETWORK ADMINISTRATION:

SOURCE INSTALLATION:

The input to this phase is the source rpm file to be installed from the server and the ip address of the clients of the network.

The processing to be done is by establishing a basic socket connectivity between the server and one of the clients using a C /C++ program written separately at the client and server and then the source rpm being transferred to the client.

The output of this phase is that the source rpm is transferred to client and the installation process is started at the using a shell script at the client.

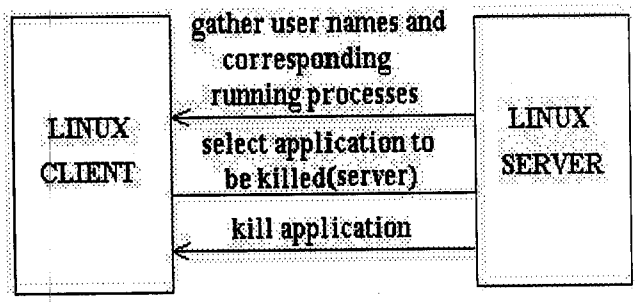


SERVICE INACTIVATION:

The input to this phase is the user name whose service is to be killed and the service name or application name that is to be killed by the administrator at the server.

The processing to be done is by establishing a basic socket connectivity between the server and one of the clients using a C /C++ program written separately at the client and server and the specified user's specified service or applications' process id is sent to the client program to kill the application or service.

The output of this phase is that the specified user's specified service or application is killed at the client.

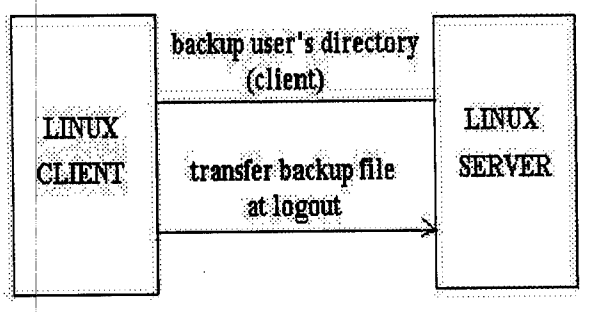


CLIENT BACKUP:

The input to this phase is the socket connectivity established at logout of client with the server.

The processing done is the backup copy of client taken at the client till log out and at logout the backup is transferred to the server.

The output of this phase is the backup copy of the user's folder available at the server.

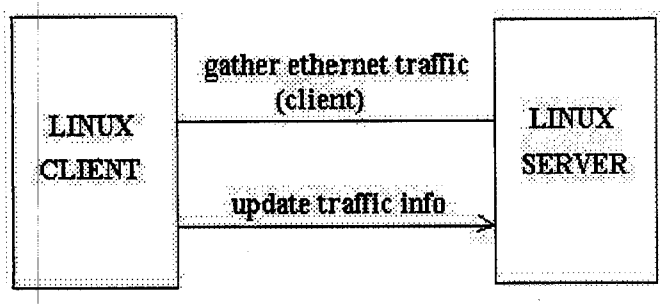


TRAFFIC ANALYSIS:

The input to this phase is the client's ip address whose Ethernet traffic is to be

The processing to be done is by establishing the basic socket connectivity between client, whose Ethernet traffic is to be measured, and server and transferring the traffic information to the server.

The output of this phase is the Ethernet traffic information of each of the clients in the server available at the server.



IMPLEMENTATION DETAILS

IMPLEMENTATION DETAILS:

PROCESSING ENVIRONMENT :

HARDWARE SPECIFICATION

Processor	Pentium III 550 Mhz and above
Floppy Disk Drive	1.44 MB
Hard Disk	20GB
Mouse	Linux compactable PS/2 mouse.
Keyboard	Multimedia Keyboard
System RAM	VGA card with onboard 8 MB V RAM supporting a solution of 1024 * 786 with 16-bit color depth running.
Monitor	Color monitor with refresh rate of 85HZ.

SOFTWARE SPECIFICATION

Languages Used	GNU C Library, Linux Programming
Operating System	Red Hat Linux 7.1.
Compilers	CC & CPP Compilers.

CLIENT MONITORING:

MESSAGING AND CHATTING:

Step1: Start

Step2: Initialize the client and server sockets with equal port number.

Step3: Initiate the socket data transfer using connect signal from the client.

Step4: Acknowledge the data transfer using the Accept signal from the server.

Step5: Send and receive data between client and server using Send, receive, sendto, receivefrom signals.

Step6: Close the socket and quit.

FILE TRANSFER:

Step1: Start

Step2: Initialize the client and server sockets with equal port number.

Step3: Initiate the socket data transfer using connect signal from the client.

Step4: Acknowledge the data transfer using the Accept signal from the server.

Step5: Request to the client from the server the file to be transferred with destination address.

Step6: Open the file to be transferred to the client in read mode and find out the number of characters in the file.

Step7: Send the number of characters to the client first.

Step8: Open a new file in client with the same name in write mode.

Step9: Transfer the contents of the required file character by character to the Client.

Step10: Receive the characters from the server and write it into the new file, repeat this loop the number of character times.

CLIENT INFORMATION:

Step1: Start

Step2: Initialize the client and server sockets with equal port number.

Step3: Initiate the socket data transfer using connect signal from the client.

Step4: Acknowledge the data transfer using the Accept signal from the server.

Step5: Gather the client information from various system files in the client and write it in a file.

Step6: Send the number of characters in the client information file to the server first.

Step7: Open a new file in server with the same name in write mode.

Step8: Transfer the contents of the required file character by character to the server.

Step9: Receive the characters from the client and write it into the new file, repeat this loop the number of character times.

Step10: Close socket and quit.

SCREEN CAPTURING:

Step1: Start

Step2: Initialize the client and server sockets with equal port number.

Step3: Initiate the socket data transfer using connect signal from the client.

Step4: Acknowledge the data transfer using the Accept signal from the server.

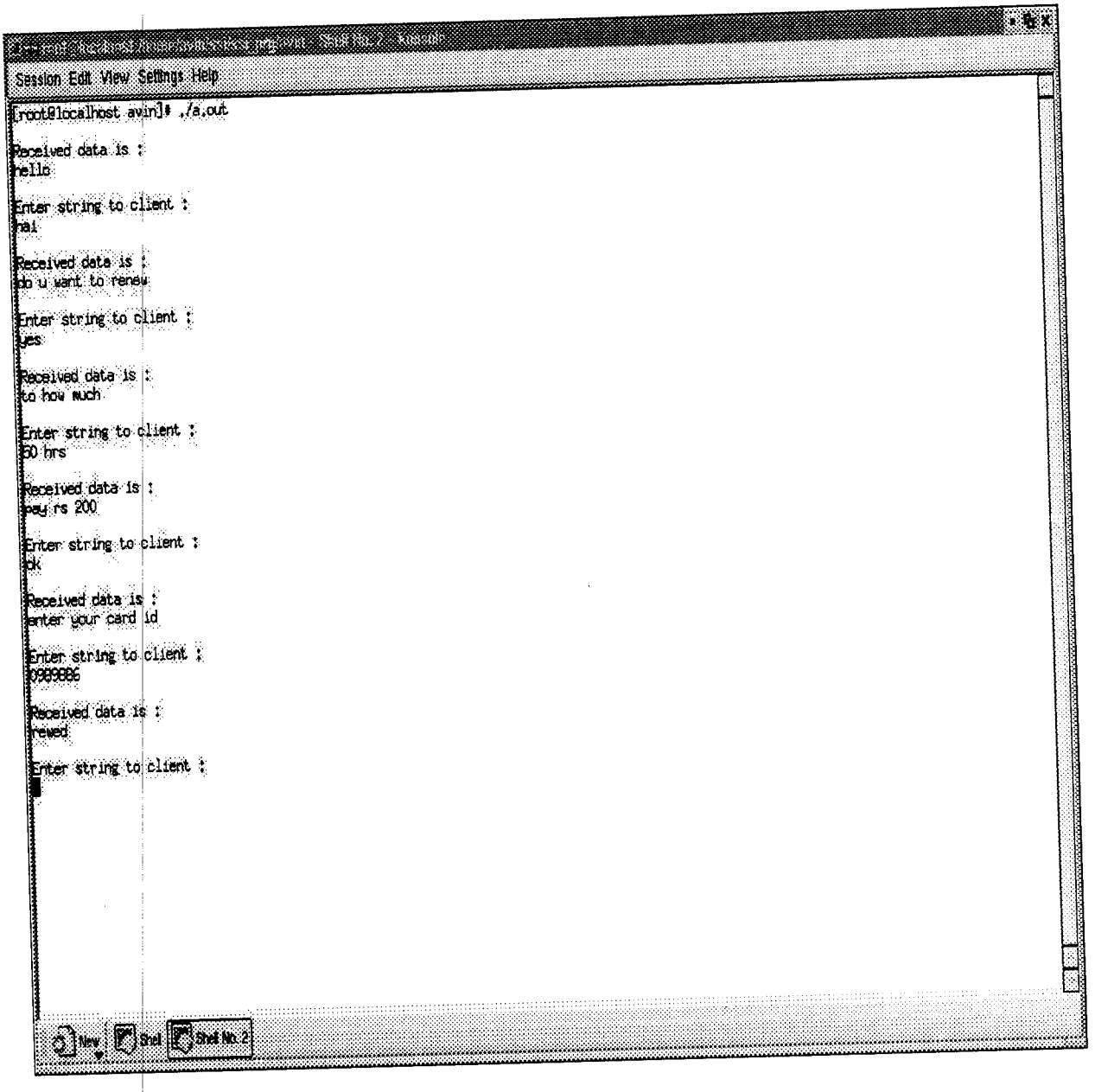
Step5: Capture the client screen using the import function and store it in a file in client.

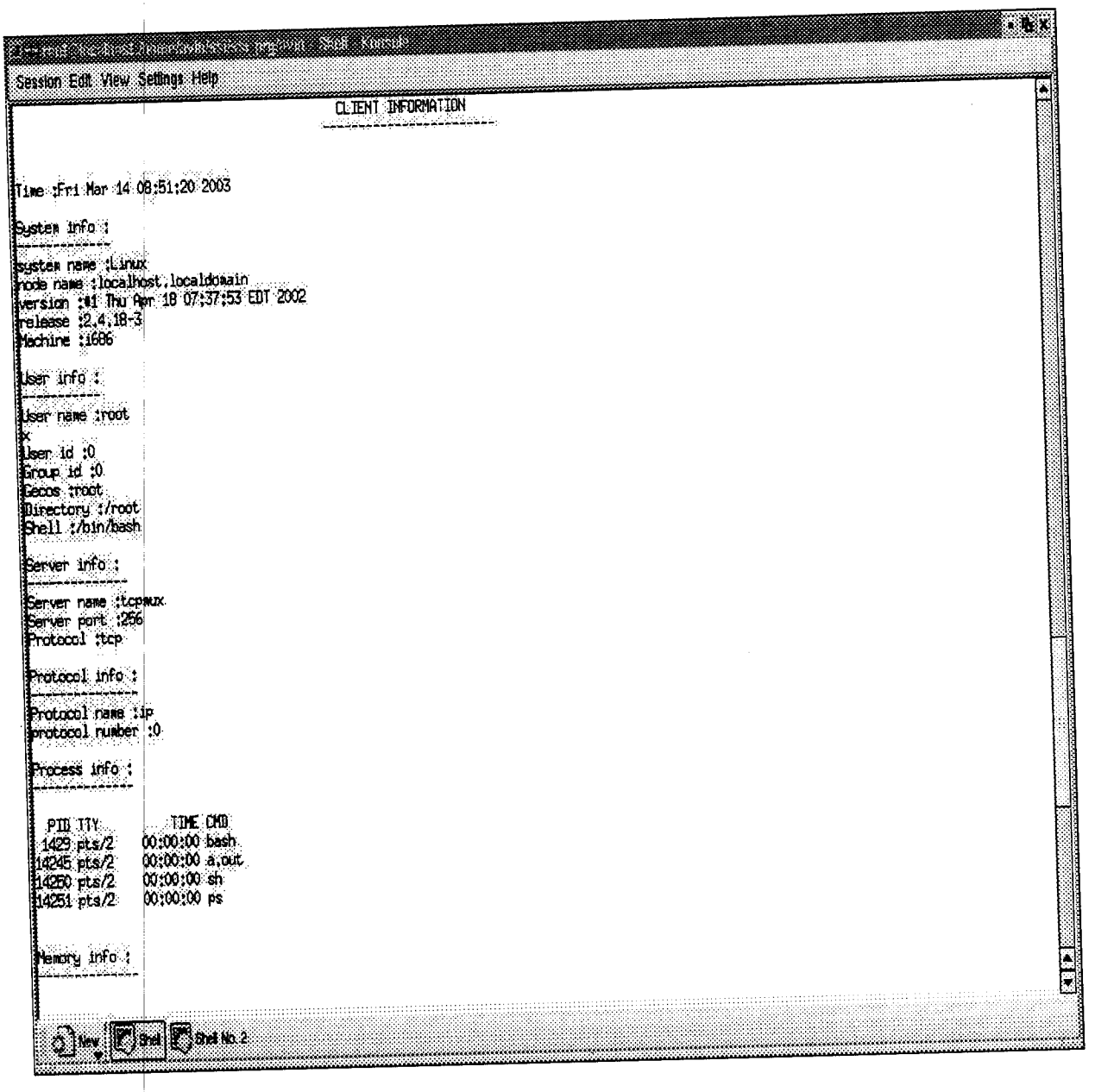
Step6: Send the number of characters in the client image file to the server first.

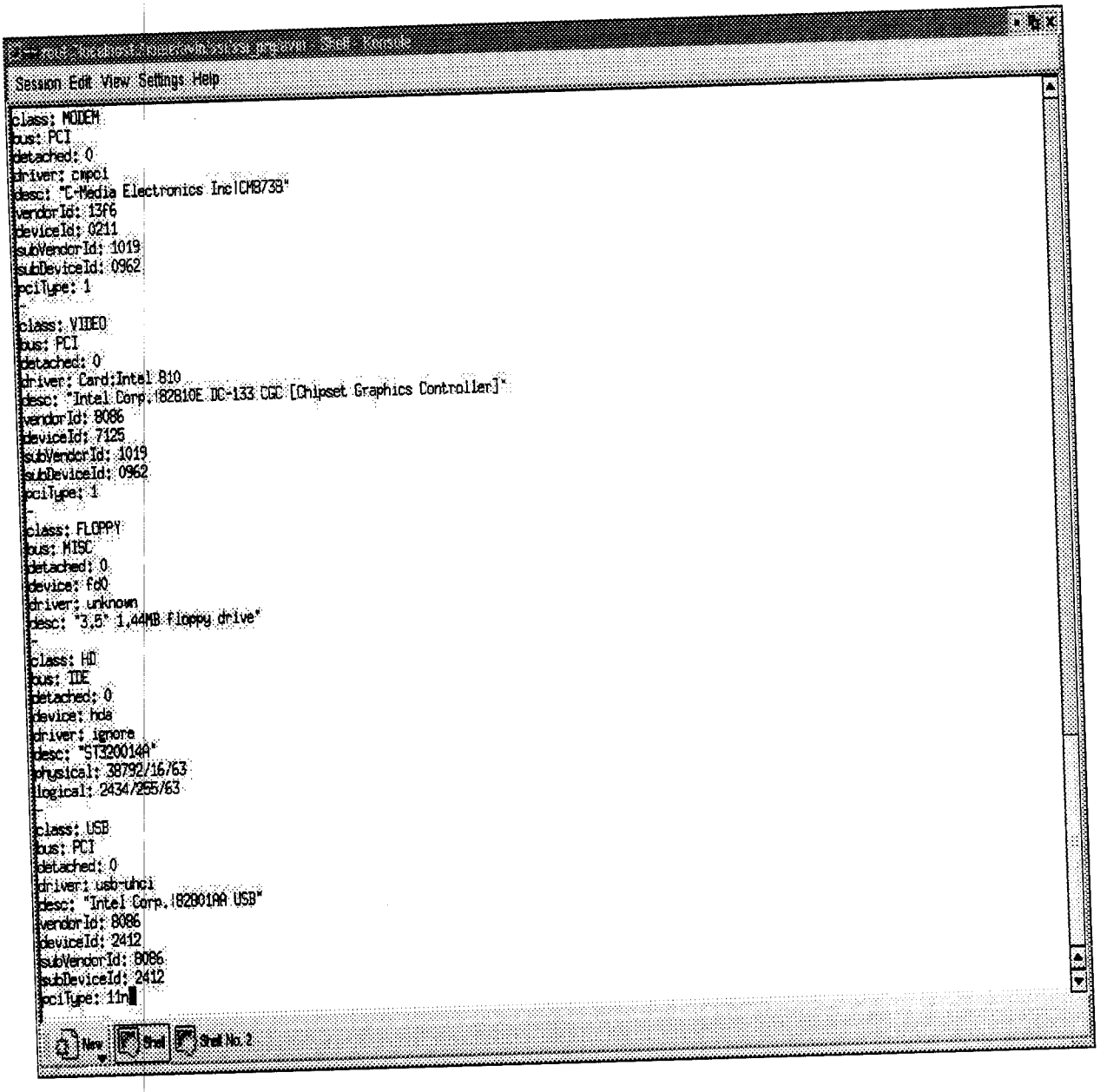
Step7: Open a new file in server with the same name in write mode.

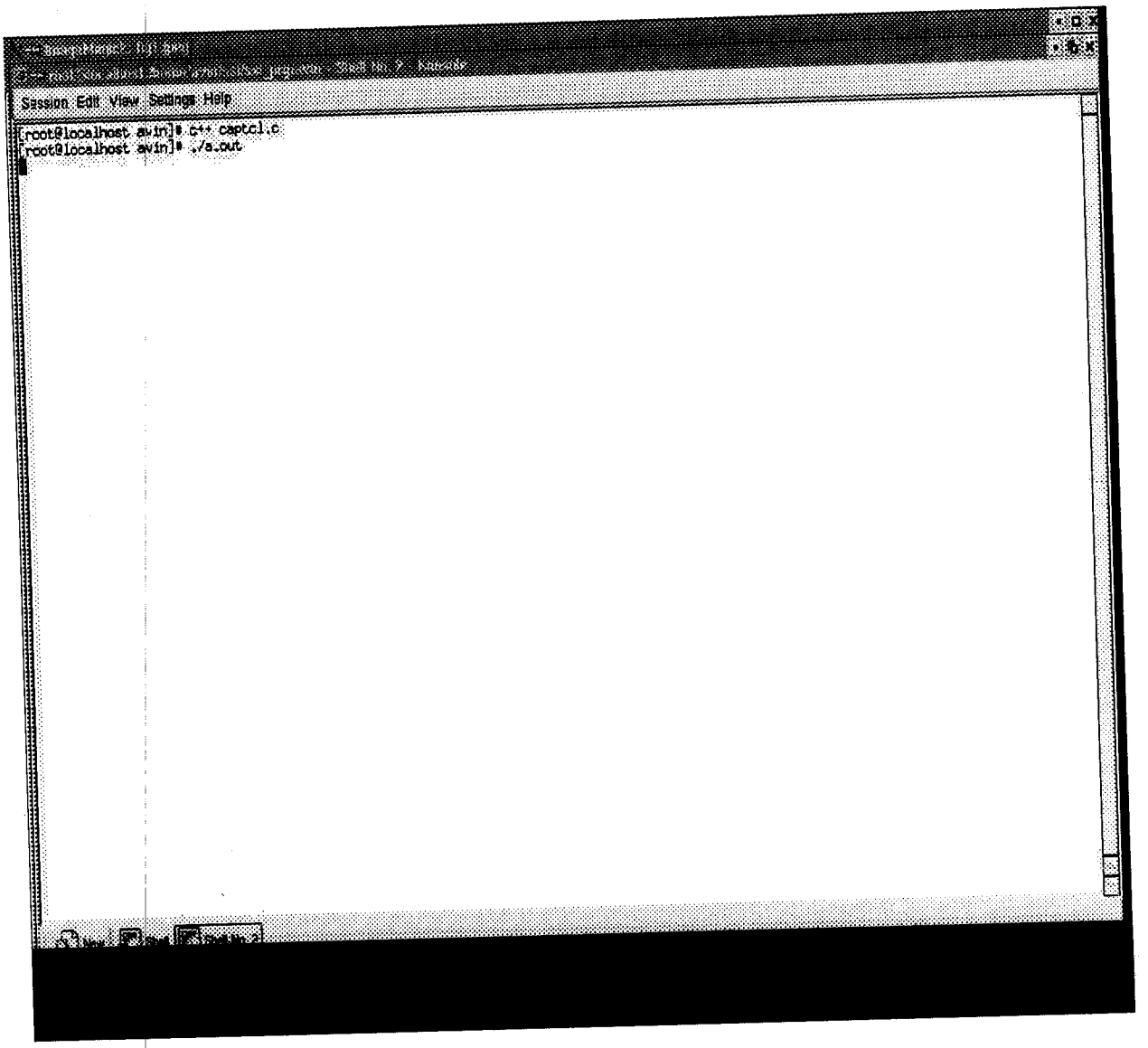
Step8: Transfer the contents of the required file character by character to the server.

Step9: Receive the characters from the client and write it into the new file, repeat this loop the number of character times.


Session Edit View Settings Help
[root@localhost avin]# ./a.out
Received data is :
hello
Enter string to client :
hai
Received data is :
do u want to renew
Enter string to client :
yes
Received data is :
to how much
Enter string to client :
50 hrs
Received data is :
pay rs 200
Enter string to client :
ok
Received data is :
enter your card id
Enter string to client :
9999886
Received data is :
rewed
Enter string to client :
New
Shell
Shell No. 2




Session Edit View Settings Help
class: MODEM
bus: PCI
detached: 0
driver: cmpci
desc: "C-Media Electronics Inc|CM8738"
vendorId: 13f6
deviceId: 0211
subVendorId: 1019
subDeviceId: 0962
pciType: 1
-
class: VIDEO
bus: PCI
detached: 0
driver: Card:Intel 810
desc: "Intel Corp.|82810E DC-133 CGC [Chipset Graphics Controller]"
vendorId: 8086
deviceId: 7125
subVendorId: 1019
subDeviceId: 0962
pciType: 1
-
class: FLOPPY
bus: MISC
detached: 0
device: fd0
driver: unknown
desc: "3.5" 1.44MB floppy drive"
-
class: HD
bus: IDE
detached: 0
device: hda
driver: ignore
desc: "ST320014A"
physical: 38792/16/63
logical: 2434/255/63
-
class: USB
bus: PCI
detached: 0
driver: usb-uhci
desc: "Intel Corp.|82801AA USB"
vendorId: 8086
deviceId: 2412
subVendorId: 8086
subDeviceId: 2412
pciType: 1
New
Shell
Shell No. 2


Session Edit View Settings Help
[root@localhost avin]# c++ captcl.c
[root@localhost avin]# ./a.out

NETWORK ADMINISTRATION MODULE:

SOURCE INSTALLATION:

Step1: Start

Step2: Create an user area in server without password by deleting its passwd structure.

Step3: Initialize the client and server sockets with equal port number.

Step4: Initiate the socket data transfer using connect signal from the client.

Step5: Acknowledge the data transfer using the Accept signal from the server.

Step6: Using FTP transfer the source RPM to all the client that requests for installation from the created new user area.

Step7: Write a shell script in the client side to run rpm –i command as soon as the source is received.

Step8: Close the socket and quit.

SERVICE INACTIVATION:

Step1: Start

Step2: Initialize the client and server sockets with equal port number.

Step3: Initiate the socket data transfer using connect signal from the client.

Step4: Acknowledge the data transfer using the Accept signal from the server.

Step5: Request from the server for all the users using the network and the node.

Step6: Select the user the server wants to control and sent the message to all the clients.

Step7: Now the client processes are stored in a file in client.

Step8: Send the number of characters in the client information file to the server

Step10: Transfer the contents of the required file character by character to the server.

Step11: Receive the characters from the client and write it into the new file, repeat this loop the number of character times.

Step12: Display the client processes in the server, check whether any service is to be stopped.

Step13: Transfer the service name to client and run the kill command in the client with the received service name and kill the process.

Step14: Close the socket and quit.

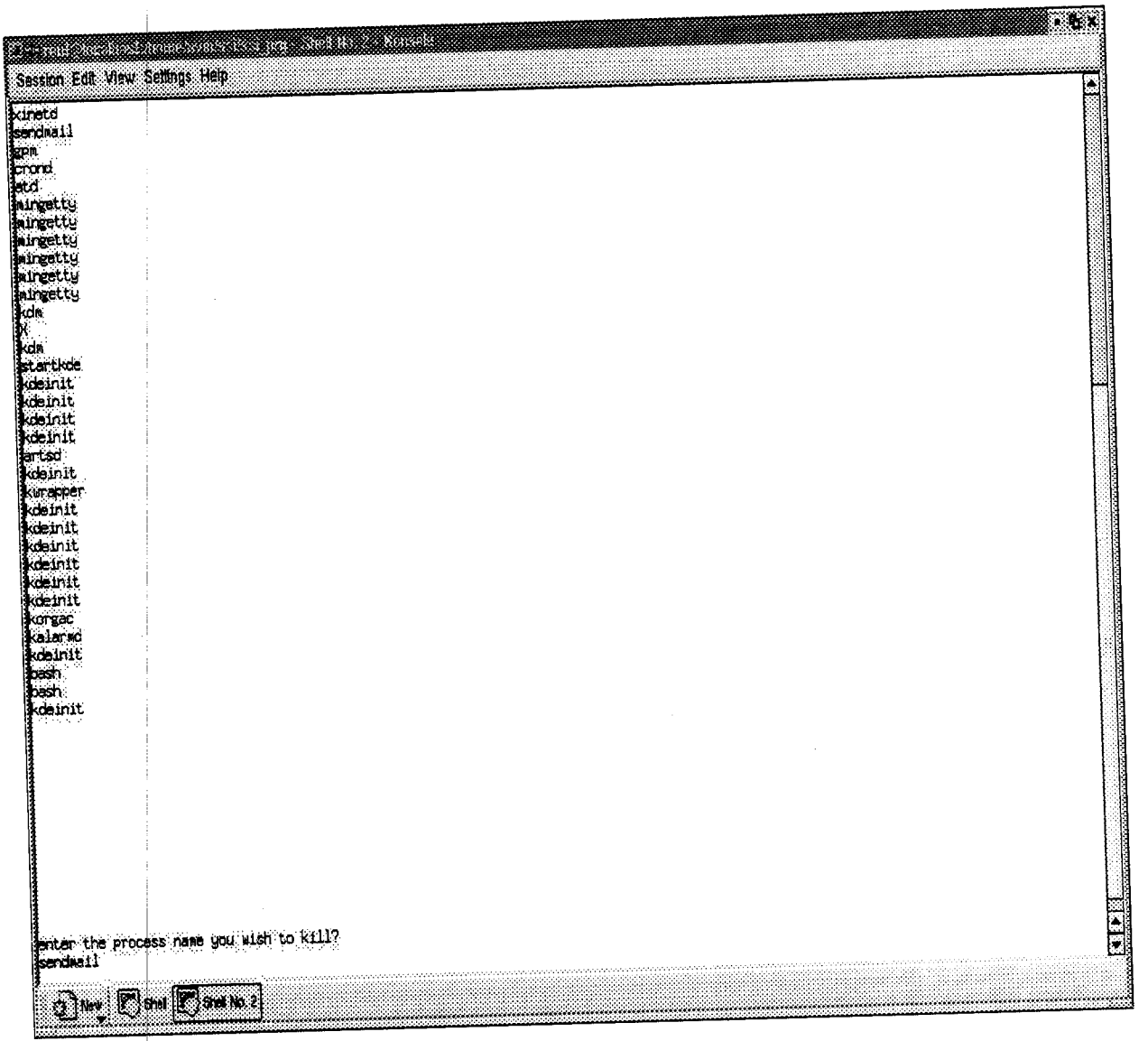
CLIENT BACK-UP:

Step1: Start

Step2: Open the .bash_logout file in all user area in client.

Step3: using tar command and setting the crontab for 5 minutes take backup of the user directory

Step4: close the .bash_logout file and quit.


Session Edit View Settings Help
xinetd
sendmail
gpm
crond
atd
mingetty
mingetty
mingetty
mingetty
mingetty
mingetty
kdm
X
kdm
startkde
kdeinit
kdeinit
kdeinit
kdeinit
artsd
kdeinit
kwrapper
kdeinit
kdeinit
kdeinit
kdeinit
kdeinit
kdeinit
korgac
kalarmd
kdeinit
bash
bash
kdeinit
enter the process name you wish to kill?
sendmail
New
Shell
Shell No. 2

TRAFFIC ANALYSIS MODULE:

ETHERNET TRAFFIC:

Step1: Start

Step2: Initialize the client and server sockets with equal port number.

Step3: Initiate the socket data transfer using connect signal from the client.

Step4: Acknowledge the data transfer using the Accept signal from the server.

Step5: Gather the client Ethernet traffic from output of ifconfig command in the client and write it in a file.

Step6: Send the number of characters in the client traffic information file to the server first.

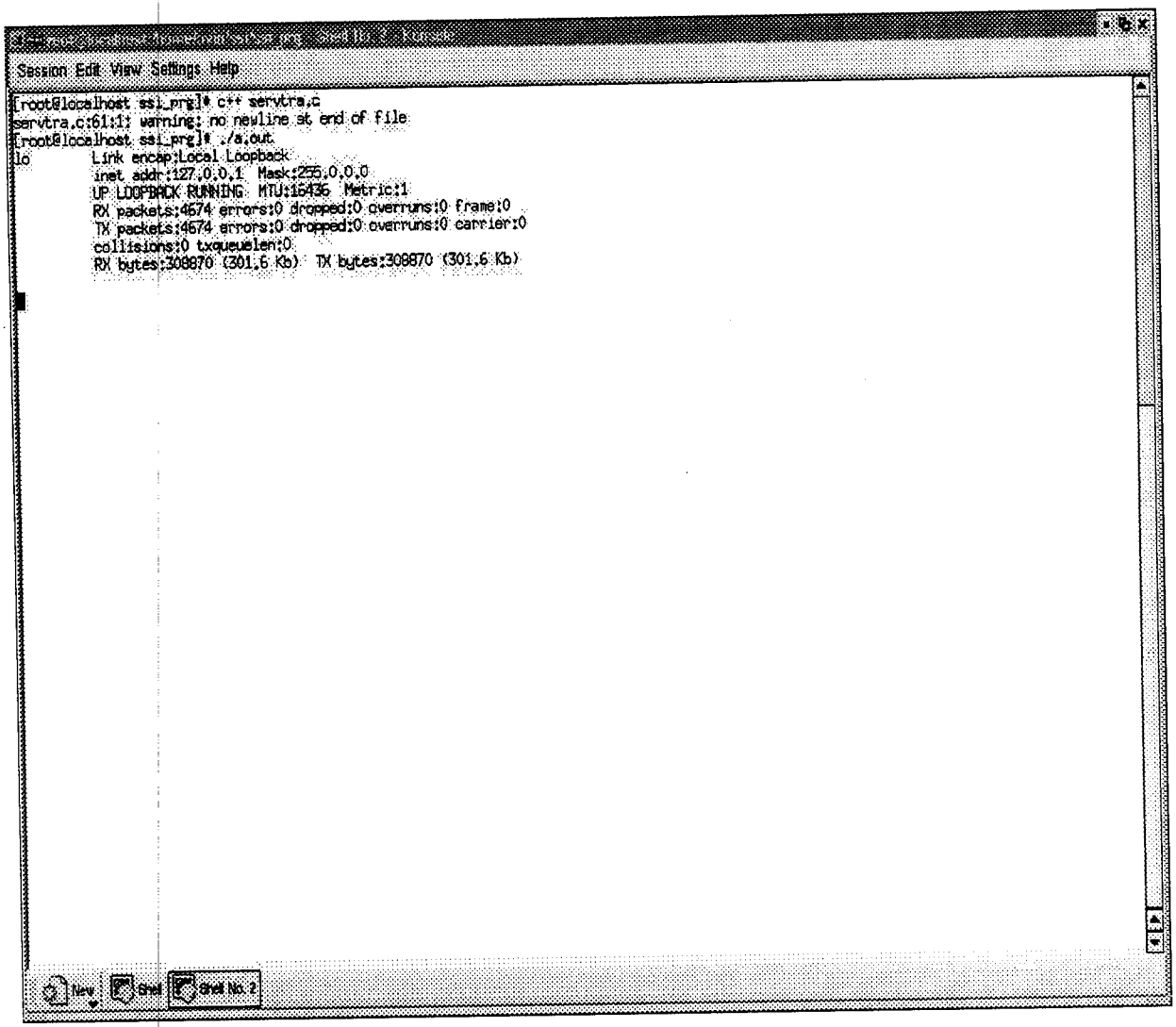
Step7: Open a new file in server with the same name in write mode.

Step8: Transfer the contents of the required file character by character to the server.

Step9: Receive the characters from the client and write it into the new file, repeat this loop the number of character times.

Step10: Display Client Ethernet traffic.

Step11: Close socket and quit.


Session Edit View Settings Help
[root@localhost ssi_prg]# c++ servtra.c
servtra.c:61:1: warning: no newline at end of file
[root@localhost ssi_prg]# ./a.out
lo Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
UP LOOPBACK RUNNING MTU:16436 Metric:1
RX packets:4674 errors:0 dropped:0 overruns:0 frame:0
TX packets:4674 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:308870 (301.6 Kb) TX bytes:308870 (301.6 Kb)
New
Shell
Shell No. 2

TEST PLAN

TEST PLAN:

1. Tests conducted on Basic connectivity between client and server:

   TEST INPUT: Socket attributes and port number.

   TEST RESULT: Connectivity established between client and server, if port number was specified correctly else a ' binding error ' is displayed and if socket attributes were specified incorrectly then core segmentation fault results.

TESTS CONDUCTED FOR NETWORK MONITORING MODULE:

2. Messaging and Chatting:

   TEST DATA: Queries and results in-between client and server.

   TEST INPUT: All alphanumeric characters.

   TEST RESULT: All characters transferred successfully between client and server.

3. File Transfer:

   TEST DATA: File information.

   TEST INPUT: File to be transferred.

   TEST RESULT: File transferred from server to client if the file specified was present in the specified directory else 'There is no such a file / directory' is displayed.

4. Client information:

TEST INPUT: The client ip address.

TEST RESULT: The client's hardware, process, network, server and other information updated to server if the specified client's ip address was correct else ' binding error ' results.

5. Screen capturing:

TEST INPUT: The client ip address.

TEST RESULT: The client's hardware, process, network, server and other information updated to server if the specified client's ip address was correct else ' binding error ' results.

TESTS CONDUCTED IN NETWORK ADMINISTRATION MODULE:

6. Source installation:

TEST DATA: Source rpm file.

TEST INPUT: The ip address' of clients.

TEST RESULT: Specified source installed in all specified clients if the specified ip address are correct otherwise ' binding error ' is displayed and if the socket attributes specified were incorrect then core segmentation fault results and if the specified source file name was incorrect ' No such file or directory' is displayed and if the login information was incorrect ' login failed ' is displayed and there will a continuous retry of establishment of connection and if the source rpm file name specified was incorrect then ' No such file/directory displayed' and finally if the login where the source rpm was present has an authentication stage along with

7. Service inactivation:

TEST DATA: Application / Service name to be killed.

TEST INPUT: User name and the corresponding service to be killed

TEST RESULT: The service specified for a particular user name is killed if the specified ip address are correct otherwise ' binding error ' is displayed and if the socket attributes specified were incorrect then core segmentation fault results and if the user name was specified incorrect then 'No such User ' error is displayed and if the service name / application name is specified incorrect then 'sh : command not found ' error results.

8. Client Backup:

TEST RESULT: Backup copy of the users directory is taken till client logs out and backup file transferred successfully to the server if the specified ip address are correct otherwise ' binding error ' is displayed and if the socket attributes specified were incorrect then core segmentation fault results.

TESTS CONDUCTED IN TRAFFIC ANALYSIS MODULE:

TEST DATA: Traffic information.

TEST RESULT: The number of packets being transferred by that Ethernet card is displayed if the specified ip address are correct otherwise ' binding error ' is displayed and if the socket attributes specified were incorrect then core segmentation fault results.

FUTURE ENHANCEMENTS

FUTURE ENHANCEMENTS:

- The future enhancements of the proposed system are to have advanced monitoring and administrations features and also include traffic analysis and control through all the devices in the network.

- We also propose to extend all the features of our system to Windows and UNIX nodes.

- We propose to set threshold alarms for all hardware and software attributes.

- We propose to store all the management details in a database 'My Sql' instead of storing it in a file.

CONCLUSION

CONCLUSION

The 'network management package' includes various monitoring/administration features along with traffic analysis features server to be a complete network management model. The administrator is made available with lot of management options to have total information of the network. The TCP/IP utility provides a elegant way with which even a layman is comfortable with.

The project was really an educative venture that provided us with the opportunity to learn a lot of Linux Programming, management concepts and system programming. An improvement in the management techniques was proposed and implemented which ensures greater efficiency. Coexistence and working together were the other lessons learnt.

Any software that does not evolve is more dead than alive. Even this product has a lot of scope for improvement. The traffic analysis can extend to all the devices on the network and the traffic control module can too be added. With a lot of advancements and sophistications in Network technology lets keep our fingers crossed for the future developments.

BIBLIOGRAPHY

BIBLIOGRAPHY

- Linux Programming – Richard Stones & Neil Matthew, Wrox Press, August 1999.

- TCP/IP Illustrated, volume 2 - Gary R. Wright & W. Richard Stevens, Addison-Wesley Publications, June 2000.

- Unix Network Programming Volume-I, W. Richard Stevens, Prentice – Hall Publications, June 1999.

- Internetworking with TCP/IP, Douglas E# Comer, David L Stevens, PHI, Feb 2001.

REFERENCES

REFERENCES:

- EXPERTS:
  - Mr. MohanRaj, WIPRO INFOTECH, Chennai.
  - Mr. Parthiban, SSI, Coimbatore.

- WEBSITES:
  - www.bitpipe.com
  - www.Redhat.com
  - www.linux.org
  - www.driverzone.com
  - www.vivisimo.com

APPENDIX

APPENDIX:

CLIENT MONITORING

CHATTING(CLIENT):

```
#include<stdio.h>
#include<string.h>
#include<malloc.h>
#include<iostream.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<netdb.h>
#include<sys/types.h>
#include<unistd.h>
int main()
{
int csoc;
char data[100];
struct sockaddr_in saddr,caddr;
while(1)
{
csoc=socket(AF_INET,SOCK_STREAM,0);
saddr.sin_family=AF_INET;
saddr.sin_port=htons(2500);
saddr.sin_addr.s_addr=INADDR_ANY;
connect(csoc,(struct sockaddr *)&saddr,sizeof(saddr));
cout<<"\nEnter data :"<<"\n";
gets(data);
send(csoc,data,sizeof(data),0);
socklen_t len=sizeof(saddr);
recvfrom(csoc,data,sizeof(data),MSG_PEEK,(struct sockaddr *)&saddr,&len);
cout<<"\n"<<"Message : " << data<<"\n";}
```

CHATTING(SERVER):

```
#include<stdio.h>
#include<malloc.h>
#include<iostream.h>
#include<sys/types.h>
#include<unistd.h>
#include<string.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<signal.h>
#include<netdb.h>
int main()
{
int csoc,ssoc;
char data[100];
socklen_t len;
struct sockaddr_in saddr,caddr,peer;
ssoc=socket(AF_INET,SOCK_STREAM,0);
saddr.sin_family=AF_INET;
saddr.sin_port=htons(2500);
saddr.sin_addr.s_addr=INADDR_ANY;
if(bind(ssoc,(struct sockaddr *)&saddr,sizeof(saddr))==-1)
{
cout<<"FAILED";
return 1;
}
listen(ssoc,5);
signal(SIGCHLD,SIG_IGN);
len=sizeof(caddr);
while(1)
```

```
if(fork()==0)
{
recv(csoc,data,sizeof(data),0);
cout<<"\nReceived data is : \n" << data<<"\n";
cout<<"\nEnter string to client :\n";
gets(data);
sendto(csoc,data,sizeof(data),MSG_DONTROUTE,(struct sockaddr
*)&caddr,sizeof(saddr));
}}
return 1;
}
```

FILE TRANSFER(CLIENT):

```
#include<stdio.h>
#include<iostream.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<netdb.h>
#include<fstream.h>
#include<string.h>
#include<stdlib.h>
int main(int argc,char *argv[])
{
FILE *fp;
int csoc,i,n;
struct sockaddr_in saddr;
struct hostent *hp;
csoc=socket(AF_INET,SOCK_STREAM,0);
hp=gethostbyname("localhost.localdomain");
saddr.sin_family=AF_INET;
saddr.sin_port=htons(2500);
```

```
{
        cout<<"No file name to transfer";
        exit(EXIT_FAILURE);
}
fp=fopen(argv[1],"r");
fseek(fp,0,2);
n=ftell(fp);
fseek(fp,0,0);
char c;
char ab[100];
strcpy(ab,argv[1]);
connect(csoc,(struct sockaddr *) &saddr,sizeof(saddr));
send(csoc,(char *) &ab,sizeof(ab),0);
send(csoc,(char *) &n,sizeof(n),0);
for(i=1;i<n;i++)
{
c=fgetc(fp);
send(csoc,(char *) &c,sizeof(c),0);
}
return 1;
}
```

FILE TRANSFER(SERVER):

```
#include<stdio.h>
#include<iostream.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<netdb.h>
#include<fstream.h>
#include<stdlib.h>
int main()
```

```
int csoc,ssoc,i;
socklen_t len;
char c;
struct sockaddr_in saddr,caddr;
ssoc=socket(AF_INET,SOCK_STREAM,0);
saddr.sin_family=AF_INET;
saddr.sin_port=htons(2500);
saddr.sin_addr.s_addr=INADDR_ANY;
if(bind(ssoc,(struct sockaddr *) &saddr,sizeof(saddr))==-1)
{
cout<<"binding error   :";
return 1;
}
listen(ssoc,5);
char fnam[200];
len=sizeof(caddr);
csoc=accept(ssoc,(struct sockaddr *) &caddr,&len);
recv(csoc,(char *) &fnam,sizeof(fnam),0);
cout<<fnam<<endl;
ft=fopen(fnam,"w");
int n;
recv(csoc,(char *) &n,sizeof(n),0);
for(i=1;i<n;i++)
{
recv(csoc,(char *)&c,sizeof(c),0);
fputc(c,ft);
}
cout<<endl<<"end";
return 1;
}
```

CLIENT INFORMATION(CLIENT):

```
#include<fstream.h>
#include<iostream.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<signal.h>
#include<netdb.h>
#include<unistd.h>
#include<sys/utsname.h>
#include<pwd.h>
#include<time.h>
#include<sys/sysinfo.h>
#include<string.h>
#include<stdlib.h>
int main()
{
FILE *fp;
struct utsname uts;
time_t tt;
struct hostent *hp;
struct passwd *pd;
uid_t uid;
gid_t gid;
int csoc,i,p,n,proto1;
pid_t pp,pp1;
char
name[12],passwd[12],gecos[12],dir[12],shell[12],sname[12],proto[12],pname[12];
struct sockaddr_in saddr,caddr;
uname(&uts);
uid=getuid();
pd=getpwuid(uid);
```

```
strcpy(passwd,pd->pw_passwd);
uid=pd->pw_uid;
gid=pd->pw_gid;
strcpy(gecos,pd->pw_gecos);
strcpy(dir,pd->pw_dir);
strcpy(shell,pd->pw_shell);
system("echo "">ps.txt");
system("echo 'Process info :'>>ps.txt");
system("echo '--------------'>>ps.txt");
system("echo "">>ps.txt");
system("ps>>ps.txt");
system("echo "">>ps.txt");
system("echo "">>ps.txt");
system("echo 'Memory info :'>>ps.txt");
system("echo '--------------'>>ps.txt");
system("echo "">>ps.txt");
system("cat /proc/meminfo>>ps.txt");
system("echo "">>ps.txt");
system("echo "">>ps.txt");
system("echo 'Networking info :'>>ps.txt");
system("echo '-----------------'>>ps.txt");
system("cat /etc/sysconfig/networking/ifcfg-lo>>ps.txt");
system("echo "">>ps.txt");
system("echo "">>ps.txt");
struct servent *s;
s=getservent();
strcpy(sname,s->s_name);
p=(s->s_port);
strcpy(proto,s->s_proto);
/*s=getservbyname("ftp","tcp");
cout<<s->s_name<<"\n";
cout<<ntohs(s->s_port)<<"\n";
```

```
cout<<s->s_name<<"\n";
cout<<s->s_proto<<"\n";
cout<<ntohs(s->s_port)<<"\n"<<"\n";*/
protoent *pro;
pro=getprotoent();
strcpy(pname,pro->p_name);
proto1=pro->p_proto;
system("echo """>>ps.txt");
system("echo """>>ps.txt");
/*pro=getprotobyname("udp");
cout<<pro->p_name<<"\n";
cout<<pro->p_proto<<"\n";
pro=getprotobynumber(6);
cout<<pro->p_name<<"\n";
cout<<pro->p_proto<<"\n";*/
hp=gethostbyname("localhost.localdomain");
csoc=socket(AF_INET,SOCK_STREAM,0);
caddr.sin_family=AF_INET;
caddr.sin_port=htons(2501);
memcpy(&(caddr.sin_addr.s_addr),hp->h_addr,hp->h_length);
fp=fopen("ps.txt","r");
fseek(fp,0,2);
n=ftell(fp);
fseek(fp,0,0);
char c;
connect(csoc,(struct sockaddr *)&caddr,sizeof(caddr));
send(csoc,(char *) &n,sizeof(n),0);
send(csoc,(char *)&uts,sizeof(uts),0);
send(csoc,(char *)&tt,sizeof(tt),0);
send(csoc,(char *)name,sizeof(name),0);
send(csoc,(char *)passwd,sizeof(passwd),0);
send(csoc,(char *)&pd->pw_uid,sizeof(pd->pw_uid),0);
```

```
send(csoc,(char *)dir,sizeof(dir),0);
send(csoc,(char *)shell,sizeof(shell),0);
send(csoc,(char *)sname,sizeof(sname),0);
send(csoc,(char *)&p,sizeof(p),0);
send(csoc,(char *)proto,sizeof(proto),0);
send(csoc,(char *)pname,sizeof(pname),0);
send(csoc,(char *)&proto1,sizeof(proto1),0);
for(i=1;i<n;i++)
{
connect(csoc,(struct sockaddr *) &saddr,sizeof(saddr));
c=fgetc(fp);
send(csoc,(char *) &c,sizeof(c),0);
}
cout<<"messege sent"<<"\n";
return 1;
}
```

SCREEN CAPTURING(CLIENT):

```
#include<stdio.h>
#include<iostream.h>
//#include<sys/stat.h>
//#include<unistd.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<netdb.h>
#include<fstream.h>
#include<string.h>
#include<stdlib.h>
int main(void)
{
FILE *fp;
```

```
struct hostent *hp;
csoc=socket(AF_INET,SOCK_STREAM,0);
hp=gethostbyname("localhost.localdomain");
saddr.sin_family=AF_INET;
saddr.sin_port=htons(2500);
memcpy(&(saddr.sin_addr),hp->h_addr,hp->h_length);
system("import -window root fig.jpeg");
fp=fopen("fig.jpeg","r");
fseek(fp,0,2);
n=ftell(fp);
cout<<n;
fseek(fp,0,0);
char c;
connect(csoc,(struct sockaddr *) &saddr,sizeof(saddr));
send(csoc,(char *) &n,sizeof(n),0);
for(i=1;i<n;i++)
{
c=fgetc(fp);
send(csoc,(char *) &c,sizeof(c),0);
}
return 1;
}
```

NETWORK ADMINISTRATION:

SOURCE INSTALLATION(CLIENT):

```
#include <iostream.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <string>
```

```
{
 int csoc;
 char data[2500] = {0};
 struct sockaddr_in saddr;
 struct hostent *hp;
 csoc=socket(AF_INET,SOCK_STREAM,0);
 hp=gethostbyname("localhost.localdomain");
 saddr.sin_family=AF_INET;
 saddr.sin_port=htons(25000);
 memcpy(&(saddr.sin_addr),hp->h_addr,hp->h_length);
 connect(csoc,(struct sockaddr *) &saddr,sizeof(saddr));
 cout<<"enter the data  :";
 recv(csoc,data,sizeof(data),0);
 cout<<data<<endl<<endl;
 string s ="./inst.sh ftp://test@127.0.0.1/"; data;
 s += data;
 s += " ";
 s += data;
 system(s.c_str());
 cout<<"Data:"<<s.c_str()<<endl;
 return 1;
 }

//INSTALLATION SCRIPT
wget $1
rpm -i $2

SOURCE INSTALLATION(SERVER):
#include <iostream.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
```

```
using namespace std;
int main()
{
 int csoc,ssoc;
 socklen_t len;
 char data[2500];
 struct sockaddr_in saddr,caddr;
 ssoc=socket(AF_INET,SOCK_STREAM,0);
 saddr.sin_family=AF_INET;
 saddr.sin_port=htons(25000);
 saddr.sin_addr.s_addr=INADDR_ANY;
 if(bind(ssoc,(struct sockaddr *) &saddr,sizeof(saddr))==-1)
 {
  cout<<"binding error  :";
  return 1;
 }
 listen(ssoc,5);
 len=sizeof(caddr);
 csoc=accept(ssoc,(struct sockaddr *) &caddr,&len);
 string s = "tetex-doc-1.0.7-47.i386.rpm";
 cout<<"Sending..."<<endl;
  send(csoc,s.c_str(),s.length(),0);
 cout<<"received data  :"<<s.c_str();
 return 1;
}
```

SERVICE INACTIVATION(CLIENT):

```
#include<curses.h>
#include<iostream.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
```

```
#include<string.h>
#include<stdio.h>
#include<stdlib.h>
int main()
{
        FILE *fp1;
        int ssoc,csoc,i,n;
        char str[100],s1[100],*s2,*s4,s3[100],s5[100],*s6;
        struct sockaddr_in saddr,caddr,ssaddr;
        struct hostent *hp;
        socklen_t len;
        csoc=socket(AF_INET,SOCK_STREAM,0);
        hp=gethostbyname("localhost.localdomain");
        saddr.sin_family=AF_INET;
        saddr.sin_port=htons(2500);
        memcpy(&(saddr.sin_addr),hp->h_addr,hp->h_length);
        connect(csoc,(struct sockaddr *) &saddr,sizeof(saddr));
        recv(csoc,(char *) &str,sizeof(str),0);
        strcpy(s1,"ps -U");
        s2=strcat(s1,str);
        strcpy(s3,"|tr -s ' '|cut -d ' ' -f5|tail +2>pro.txt");
        s4=strcat(s2,s3);
        system(s4);
        fp1 = fopen("pro.txt","r");
        fseek(fp1,0,2);
        n=ftell(fp1);
        fseek(fp1,0,0);
        char c[n];
        connect(csoc,(struct sockaddr *) &saddr,sizeof(saddr));
        send(csoc,(char *) &n,sizeof(n),0);
        for(i=0;i<n;i++)
        {
```

```
        send(csoc,(char *) &c[i],sizeof(c[i]),0);
        }
      char st1r[100],s11[100],*s21,*s41,s31[100];
      recv(csoc,(char *) &st1r,sizeof(st1r),0);
      strcpy(s11,"ps -C");
      s21=strcat(s11,st1r);
      strcpy(s31,"|tr -s ' '|cut -d ' ' -f2|tail +2>pro1.txt");
      system("cat pro1.txt");
      s41=strcat(s21,s31);
      system(s41);
      fclose(fp1);
      fp1=fopen("pro1.txt","r");
       fseek(fp1,0,2);
       n=ftell(fp1);
      fseek(fp1,0,0);
       char a[n];
       for(i=0;i<n;i++)
      a[i] = fgetc(fp1);
      strcpy(s5,"kill ");
      s6=strcat(s5,a);
      system(s6);
      return 1;
}
```

SERVICE INACTIVATION(SERVER):

```
#include<iostream.h>
#include<fstream.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<netdb.h>
#include<stdio.h>
```

```
        system("cat anukill.txt");
        cout<<endl<<"enter the process name you wish to kill?"<<endl;
        cin>>str;
        connect(csoc,(struct sockaddr *)&saddr,sizeof(saddr));
        send(csoc,(char *) &str,sizeof(str),0);
        return 1;
    }
```

TRAFFIC ANALYSIS:

ETHERNET TRAFFIC(CLIENT):

```
#include<stdio.h>
#include<malloc.h>
#include<fstream.h>
#include<iostream.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<signal.h>
#include<netdb.h>
#include<unistd.h>
#include<sys/utsname.h>
#include<pwd.h>
#include<time.h>
#include<sys/sysinfo.h>
#include<string.h>
#include<stdlib.h>
int main()
{
FILE *fp;
struct hostent *hp;
int csoc,i,n;
```

```
fputc(c,ft);
}
fclose(ft);
system("cat tra1.txt");
return 1;
}
}
}
```

```
#include<pwd.h>
#include<time.h>
#include<sys/sysinfo.h>
#include<string.h>
#include<stdlib.h>
int main()
{
FILE *ft;
char c;
int csoc,ssoc,i,n;
socklen_t len;
struct sockaddr_in saddr,caddr;
ssoc=socket(AF_INET,SOCK_STREAM,0);
saddr.sin_family=AF_INET;
saddr.sin_port=htons(3501);
saddr.sin_addr.s_addr=INADDR_ANY;
if(bind(ssoc,(struct sockaddr *)&saddr,sizeof(saddr))==-1)
{
cout<<"failed"<<"\n";
return 1;
}
listen(ssoc,5);
len=sizeof(caddr);
ft=fopen("tra1.txt","w");
while(1)
{
csoc=accept(ssoc,(struct sockaddr *)&caddr,&len);
if(fork()==0)
{
recv(csoc,(char *) &n,sizeof(n),0);
system("echo ">dev1.txt");
for(i=0;i<n;i++)
```