INDEX

INDEX

Expt. No.: 5 Date: 26/05/22

IMPLEMENT A CONCURRENT TIME SERVER USING UDP AS TRASPORT LAYER PROTOCOL BY EXECUTING THE PROGRAM AT REMOTE SERVER.

AIM

To implement a concurrent time server using UDP as transport layer protocol by executing the program at remote server. Client sends a time request to server and server sends its system time back to the client. Client displays the result.

ALGORITHM

UDP Client Server

1. Create a socket for UDP using the function call, socket(AF_INET, SOCK_DGRAM, 0);
2. Declare a time object variable ct of data type, time_t.
3. The bzero() function places null bytes of memory area pointed to by local. bzero((char*)& servaddr, sizeof(servaddr));
4. Initialise the structure sockaddr_in members of sin_family, sin_addr, sin_port.
5. Bind the socket to its port using bind(s, (struct sockaddr*) & servaddr, size of(servaddr));
6. Receive time request from client recvfrom(s, buffer, 1024, 0, (struct sockaddr*) & cliaddr, &t)
7. Initialises ct = time (NULL) and prints the current date and time by calling ctime (&ct)
8. Child process is created. Parent process stops listening for new connections. Child will continue to accept TIME requests from other clients, since it is a concurrent server. The main (parent) process now handles

the connected client.

9. After clearing the buffer memory area using memset() function, TIME request is received from client using recvfrom(s, buffer, 1024, 0(struct sockaddr*) &cliaddr, &l)

10. Prints the formatted string TIME to buffer.

11. Sends back UPDATED CURRENT TIME to client using sendto(s, buffer, sizeof(buffer), 0, (struct sockaddr*) &cliaddr, sizeof(cliaddr)

12. Close the socket using close(int sockfd) function.

UDP Client.

1. Create a socket for UDP using the function call, socket(AF_INET, SOCK_DGRAM, 0);

2. The bzero() function places null bytes of memory area pointed to by local. bzero((char*) &local, sizeof(local));

3. Initialise the structure sockaddr_in members of sin_family, sin_addr, sin_port.

4. Bind the socket to its port using bind(s,(struct sockaddr*) &local, sizeof(local))

5. The bzero() function places null bytes of memory area pointed to by servaddr. bzero((char*) &servaddr, sizeof(local))

6. Client sends TIME request to server using sendto(s, buffer, sizeof(buffer), 0, (struct sockaddr*) &servaddr, sizeof(servaddr)

7. Client receives TIME response from server using recvfrom() function as follows:
recvfrom(s, buffer, 1024, 0(struct sockaddr*) &servaddr, &l)

8. Prints the received message in clients terminal.

RESULT

Implemented a concurrent time server using UDP as transport layer protocol by executing the program at remote server.

Jai
27/5/22

Expt. No.: 6 Date: 31/05/22

Page No.: 11

Multiuser Chat Server And Client

AIM

To implement a multi user chat server and client using TCP as transport layer protocol

ALGORITHM

TCP SERVER

1. Create a socket for TCP using the function call, socket((AF_INET, SOCK_STREAM, 0);
2. The memset() function fills the first n bytes of memory area pointed to by addr with constant byte 0.
3. Intialise the structure sockaddr_in members of sin_family sin_addr, sin_port.
4. Bind the socket to its port using bind (int sockid, (struct sockaddr*) & ser_addr, sizeof(ser_addr)
5. Listen for any active client connections using listen(int sockfd, int backlog);
6. Server infinetly accepts client connections using accept function call as follows:
   accept (int sockfd, (struct sockaddr*) & cl_addr, & sizeof(cladr))
7. After accepting client connection, inet-ntop() function is used to convert client network address structure src in the address of the family into a character string. The the resulting string is copied to the buffer pointed by dst, which must be a non-null pointer. The caller specifies the number of bytes available in this buffer in argument size.

```
#include <arpa/inet.h>
const char * inet_ntop (int af, const void * src, char * dst
                                          socklen_t size);
```

Teacher's Signature

8. Child process is created. Parent process stops listening for new connections. Child will continue to listen. The main (parent) process now handles the connected client.

9. After clearing the buffer memory area using memset() function, data is received from client using recv(int sockfd, void * buffer, BUF_SIZE, unsigned int flags);

10. Sends back received data to client using send(int sockfd, void * buffer, BUF_SIZE, unsigned int flag) function.

11. prints to which client IP address data was sent.

12. Close the socket using close (int sockfd) function.

TCP CLIENT

1. Create a socket for TCP using the function call, socket (AF_INET, SOCK_STREAM, 0);

2. The memset() function fills the first n bytes of memory area pointed by addr with constant byte 0.

3. Initialise the structure sockaddr_in members of sin_family, sin_addr, sin_port.

4. Connect using function connect (int sockfd, (struct sockaddr*) & ser_addr, size of (ser_addr));

5. Client reads in the line and make sure it was successful by processing the line using fgets() function infinitely in a while loop as follows

while (fgets (buffer, BUF_SIZE, stdin) != NULL))

6. Client sends data to server using send (int sockfd, void * buffer, BUF_SIZE, unsigned int flags) function.

7. Client receives response from server using recv() function as follows:

Expt. No.: ....

Date: ....

recv (int sockfd, void * buffer, BUF_SIZE, unsigned int flags)

8. prints the received message in clients terminal

9. client can continue sending messages to server, as long as server is listening.

RESULT

Successfully implemented a multiuser chat server and client using TCP as transport layer protocol.

Jai
1/6/22

Teacher's Signature ____________

M . P r a D .

Date: 14/06/22

Expt. No.: 7 (1)

Page No.: 14

SLIDING WINDOW PROTOCOLS

a) AIM

To implement stop and wait ARQ flow control protocol.

ALGORITHM

1. Start the program
2. Generate a random number that gives the total number of frames to be transmitted.
3. Transmit the first frame.
4. Receive the acknowledgement for the first frame
5. Transmit the next frame.
6. Find the remaining frames to be sent.
7. If an acknowledgement is not received for a particular frame, retransmit that frame alone again
8. Repeat the steps 5 to 7 till the number of remaining frames to be sent becomes zero.
9. Stop the program.

Teacher's Signature ____________

Expt. No.: ____________ Date: ________

RESULT

Successfully implemented stop and wait ARQ flow control protocol.

15/12/22

Teacher's Signature ____________

Expt. No.: 7 (2)

Date: 14/06/22

b) AIM : To implement Go-Back-N ARQ flow control protocol.

ALGORITHM - Sender

```
$S_w \leftarrow 2^m - 1$
 $S_f = S_n = 0$
while True do
        Wait for event()
        if Event (Request To Send) then
        if $S_n - S_f \geq S_w$ then
            Sleep()
        endif
        Get Data()
        Make Frame ($S_n$)
        Store Frame ($S_n$)
        Send Frame ($S_n$)
        $S_n \leftarrow (S_n+1)$ % $S_w$
        if timer is not running then
            Start timer()
        endif
        if Event (Arrival Notification) then
            Receive (Ack)
            if corrupted (Ack) then
                Sleep ()
            endif
          if ackNo > $S_f$ and ackNo <= $S_n$ then
              while $S_f$ <= ackNo do
                  PurgeFrame ($S_n$)
              $S_f \leftarrow (S_f+1)$ % $S_w$
              end while.
```

Teacher's Signature

```
      end if
      Stop Timer()
    end if
    if event (timeout) then
      Start Timer()
      temp ← Sf
      while temp < Sn do
          Send Frame (Sn)
          Sf ← (Sf+1) % Sw
      end while
    end if
  end while
```

RECEIVER

```
Rn ← 0
while True do
      Wait for Event()
      if Event (Arrival Notification) then
          Receiver (Frame)
          if corrupted (Frame) then
              Sleep()
          end if
          if seq No == Rn then
              Deliever Data()
              Rn ← (Rn+1) % 2^m
          end if
          Send Ack (Rn)
      end if
   END while.
```

Date: 14/06/22

Expt. No.: 7(c) Page No.: 18

c) AIM : To implement selective repeat ARQ flow control protocol

ALGORITHM - Selective Repeat ARQ Sender

```
$S_w \leftarrow 2^{m-1}$
$S_f = S_n = 0$
while True do
        Wait for Event()
        if Event(Request To Send) then
            if $S_n - S_f >= S_w$ then
                Sleep()
            end if
            Get Data()
            Make Frame($S_n$)
            Store Frame($S_n$)
            Send Frame($S_n$)
            $S_n \leftarrow (S_n+1)$ % $S_w$
            Start Timer($S_n$)
        end if
        if Event(Arrival Notification) then
            Receive(Frame)
            if corrupted(Frame) then
                Sleep()
            end if
            if frametype == NAK then
                if NakNo in [$S_f$, $S_n$] then
                    Resend(NakNo)
                    Start timer(NakNo)
                end if
            else if FrameType == Ack then
```

Teacher's Signature ________

```
            if ackNo in [Sf, Sn] then
                while Sf < ackNo do
                    Purge (Sf)
                    Stop Timer (Sf)
                    $S_f \leftarrow (S_f+1) \% 2^m$
                end while
            end if
        end if
        end if
    if Event ( Time Out Ti ) then
        Start Timer ( Ti )
        Send Frame ( Ti )
    end if
    end while.
```

SELECTIVE REPEAT RECEIVER

```
Rn ← 0
nak sent ← false
ack needed ← false
for all slots in slots do
    Marked (slot) ← false
end for.
while True do
    Wait for Event()
    if Event (Arrival Notification) then
        Receive ( Frame)
        if corrupted ( Frame ) and not naksent then
            Sent NAK ( Rn)
            naksent ← True
            Sleep()
```

Teacher's Signature ____________________

```
        endif
        if SeqNo ! = Rn and not a nakSent then
            SendNAk (Rn)
            nakSent ← True
            if seqno in window and not marked (seqno) then
                StoreFrame (seqno)
                Marked (seqno) ← True
                while Marked (Rn) do
                    Delieuer Data (Rn)
                    Purge (Rn)
                    Rn ← (Rn+1) % $2^m$
                    ackNeeded ← True
                end while
                if ackNeeded then
                    Send Ack (Rn)
                    ackNeeded ← False
                    nakSent ← False
                end if
            end if
          end if
        end if
    end while.
```

RESULT

15/6/22

Successfully implemented selective repeat ARQ flow control protocol.

Expt. No.: 8

Date: 04/07/22

Page No.: 21

LINK STATE ROUTING

AIM

To implement and simulate link state protocol.

ALGORITHM

1. D(v) : cost of the least cost path from the source node to the destination node v as of this iteration of the algorithm.
2. P(v) : previous node of v along the current least cost path from source to v.
3. N' : subset of nodes. v is in N' if the least cost path from the source to v is definitely known.
4. Initialisation:

```
N' = {u}
for all nodes v
    if v is a neighbour of u
      then D(v) = c(u,v)
    else
    D(v) = ∞
 do {
    find w not in N' such that D(w) is a minimum
    add w to N'
    update D(v) for each neighbour v of w and not in N'
    D(v) = min { D(v), D(w) + c(w,v)}
  }
while (N' != N)
```

RESULT

Successfully implemented and simulated link state protocol.

Teacher's Signature

M P & D D.

Date: 11/07/22

Expt. No.: 9 Page No.: 22

CONCURRENT FTP

AIM

Program to implement concurrent FTP server and client for file transfer to server.

ALGORITHM . SERVER

1. Create a socket using socket() system call with address family AF_INET, Type SOCK_STREAM and default protocol.
2. Initialize address structure with NULL assign port number and IP address to the socket created.
3. Bind server address and port using bind() system call by binding the socket id with the socket structure.
4. Listen for active TCP connections (upto 10) in the socket file descriptor.
5. Wait for the client connection to complete accepting connections using accept() system call.
6. Display information of connected client and print the number of clients connected till now.
7. Create a new child process for each client using fork() system call.
8. Receive the client file using recv() system call.
9. Using *fgets (char *str, int n, FILE *stream) function, we read a line of text from the specified stream and stores it into the string pointed to by str. It stops when either (n-1) characters are read, or when the end of file is reached 10. On successful execution i.e) when file pointer reaches end of file, file transfer

Teacher's Signature

"completed" message is sent by the server to the accepted client connection using newsd, socket file descriptor.

CLIENT

1. Create a socket system call with address family, AF_INET, type SOCK_STREAM and default protocol.
2. Initialise address structure with NULL, assign port number and IP address to the socket created.
3. Enter the client port id.
4. Connect to the server address using connect() system call.
5. Read the existing and new file name from user.
6. Send existing file to server using send() system call.
7. Receive feedback from server "completed" regarding file transfer completion.
8. Display the message, in the file on the client screen.
9. Write 'file is transferred' message to standard output screen of client and exit.
10. Close the socket communication.

RESULT

12/3/22

Successfully implemented concurrent FTP server and client for file transfer to server.

LEAKY BUCKET

AIM

To implement congestion control using leaky bucket algorithm.

ALGORITHM.

1. Start
2. Set the bucket size or the bigger size.
3. Set the output rate
4. Transmit the packets such that there is no overflow
5. Repeat the process of transmission until all packets are transmitted.
6. Stop.

RESULT

Successfully implemented congestion control using leaky bucket algorithm.

12/07/22

Date: 17/07/22

Expt. No.: 11

Page No.: 25

Study Of Wireshark Tool

AIM

To study the working of wireshark tool.

Wireshark has a very rich history. Gerald Combs, a computer science graduate of the university of missouri at kansas city originally developed it out of necessity. The first version of Comb's application called Ethereal, was released in 1998 under the GNU public License (GPL). Eight years after releasing Ethereal, Combs left his job to pursue other career opportunities. Unfortunately, his employer at that time had full rights to Ethereal trademarks and Combs was unable to reach an aggrement that would allow him to control the Ethereal "brand". Instead Combs and the rest of the development team rebranded the project as wireshark in mid-2006 thereafter it continued.

The Benefits Of Wireshark

- Supported Protocols: Wireshark excels in the number of protocols that it supports more than 850 as of this writing. These range from common ones like ip and DHCP to more advanced proprietary protocols like Apple Talk and Bit Torrent.
- User friendliness: The wireshark interface is one of the easiest to understand of any packet sniffing application. It is a GUI-based with very clearly written context

Teacher's Signature

menus and a straightforward.

Layout: It also provides several features designed to enhance usability, such as protocol based color coding and detailed graphical representation of raw data. Unlike some of the more complicated command line driven alternatives, like tcp dump, the wireshark GUI is great for those who are just entering the world of packet analysis.

Cost: Since it is open source, wiresharks pricing can't beat: Wire-shark is released as free software under the GPL.

Program Support: A software package's level of support can make or break it. When dealing with freely distributed software such as wireshark, there may not be any formal support, which is why the open source community often relies on its user base to provide support.

Operating Support System: Wireshark supports all major modern operating systems, including window, Mac OS, and linux-based platforms.

OBJECTIVE:

- Use Wireshark to monitor an ethernet interface for recording packet flows.
- Generate a TCP connection using a web browser
- Observe the initial TCP/IP three-way handshake.

RESULT.

Successfully studied the working of wireshark tool.

Jai 20/7/22

STIMULATING NS2 SIMULATOR

AIM

To install network simulator NS2 in any of the linux operating system stimulate wired and wireless scenerios.

DESCRIPTION

A simulation can be thought of as a flow process of network entity (eg nodes, packet) is these entities move through system they interact with other entities. Join certain activities trigger events cause some changes to the state of the system and leave the process from time to time. They contend or wait for some type of resources. This implies that there must be a logical execution sequence to cause all these actions to happen in a comprehensible and managable way.

INTRODUCTION TO NETWORK SIMULATOR 2 (NS2)

Network simulator (version 2) widely known as NS2 is simply an event driven simulation tool that has proved useful in studying the dynamic nature of communication networks simulation of wired as well as wireless network functions and protocols (eg routing algorithm, TCP, UDP) can be done using NS2 in general NS2 provides users with a way of specifying such network protocols and simulating corresponding behaviours.

BASIC ARCHITECTURE

NS2 provides uses with an executable command or which takes on input argument. The name of TCL simulation scripting like in most cases simulation trace like is created and used to photograph and or to create animation NS2 consist of two key languages C++ and object oriented, tools command language while the C++ defines the internal mechanism of the simulation object and setup simulation by assembling and configuring the object as well as scheduling discrete events (ie frontend)

RESULT

Successfully studied the working of NS2 simulator.

27/7/22