Experiment Name / No.: MULTIUSER CHAT SERVER AND CLIENT

Camlin | Page No. | Date / /

AIM: To implement a multi user chat server 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. Initialize the structure Sockaddr_in members of sin_family, sin_addr, sin_port.
4. Bind the socket to its port using bind (int sockfd, (struct sockaddr*) &ser_addr, sizeof (ser_addr);
5. Listen for any active client connections using listen (int sockfd, int backlog); Backlog argument defines the maximum length to which queue of pending connections for sockfd may grow.
6. Server infinitely accepts client connections using accept function call as follows:
accept (int sockfd, (struct sockaddr*) &cl_addr, &sizeof (cl_addr);
7. After accepting client connection, ineb_ntop() function is used to convert client network address structure src in the af address family into a character string. 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);
8. Child process is created. parent process stops listening for.

Teacher's Signature:

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.

Algorithm - 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. Initialize the structure sockaddr_in members of sin_family, sin_addr, sin_port
4. connect using function connect (int sockfd, (struct sockaddr*) &ser_addr, sizeof (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 a recv() function as follows:
recv (int sockfd, void *buffer, BUF_SIZE, unsigned int flags);
8. prints the received message in client's terminal.

Teacher's Signature:

Experiment Name / No.:

Camlin | Page No.
Date / /

9. Client can continue Sending Messages to Server, as long as Server is listening.

RESULT:

Teacher's Signature:

Experiment Name / No.: Sliding Window Protocols

Camlin | Page No.
Date / /

(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.

Experiment Name / No.: _____

Camlin | Page No.

Date / /

RESULT :

Experiment Name / No.:

Page No.

Date / /

(b) AIM: Implement Go-Back-N ARQ flow control protocol.

Algorithm: GoBack-N Protocol - Sender

```
1.  Sw ← 2^m - 1
2.  Sf = Sn = 0
3.  While (true) do
4.      WaitForEvent()
5.      If Event (RequestToSend) then
6.          If Sn - Sf ≥ Sw then
7.              Sleep()
8.          endif
9.          GetData()
10.         MakeFrame(Sn)
11.         StoreFrame(Sn)
12.         SendFrame(Sn)
13.         Sn ← (Sn + 1) % Sw
14.         If Timer is not running then
15.             StartTimer()
16.         endif
17.     If Event (ArrivalNotification) then
18.         Receive (ACK)
19.         If Corrupted (ACK) then
20.             Sleep()
21.         endif
22.         If ackNo > Sf and ackNo <= Sn then
23.             While Sf <= ackNo do
24.                 PurgeFrame(Sn)
25.                 Sf ← (Sf + 1) % Sw
```

Teacher's Signature:

```
        endwhile
      endif
      StopTimer()
    endif
    if Event(Timeout) then
      StartTimer()
      temp ← Sf
      while temp < Sn do
        SendFrame(Sn)
        Sf ← (Sf + 1) % Sw
      endwhile
    endif
  endwhile
```

Algorithm GoBack-N Receiver

```
1. Rn ← 0
2. while true do
3.   WaitForEvent()
4.   if Event(ArrivalNotification) then
5.     Receive(Frame)
6.     if Corrupted(frame) then
7.       Sleep()
8.     endif
9.     if SeqNo == Rn then
10.      DeliverData()
11.      Rn ← (Rn + 1) % 2^m
12.    endif
13.    SendAck(Rn)
14.  endif
15. End while
```

RESULT:

Experiment Name / No.: Page No.

Date / /

(c) AIM: To Implement Selective Repeat ARQ flow control protocol.

Algorithm: Selective Repeat ARQ - Sender

```
1.  Sw ← 2^(m-1)
2.  Sf = Sn = 0
3.  While True do
4.      WaitForEvent()
5.      If Event (RequestToSend) then
6.          If Sn - Sf >= Sw then
7.              Sleep()
8.          endif
9.          GetData()
10.         MakeFrame (Sn)
11.         StoreFrame (Sn)
12.         SendFrame (Sn)
13.         Sn ← (Sn + 1) % Sw
14.         StartTimer (Sn)
15.     endif
16.     If Event (ArrivalNotification) then
17.       Receive (frame)
18.       If Corrupted (frame) then
19.         Sleep()
20.       endif
21.       If FrameType == NAK then
22.         If nakNo in (Sf, Sn] then
23.             Resend (nakNo)
24.             StartTimer (nakNo)
25.         endif.
```

Teacher's Signature:

Experiment Name / No.: _____ Camlin Page No. Date / /

```
26.      else if FrameType == Ack then
27.        if ackNo in (Sf, Sn], then
28.          while Sf < ackNo do
29.            Purge (Sf)
30.            StopTimer (Sf)
31.            Sf ← (Sf + 1) % 2^m
32.          endwhile.
33.        endif
34.      endif
35.    endif
36.    if Event (TimeOut Ti) then
37.      StartTimer (Ti)
38.      SendFrame (Ti)
39.    endif
40.  endwhile
```

Algorithm : Selective Repeat - Receiver

```
1.  Rn ← 0
2.  naksent ← false
3.  ackNeeded ← false
4.  for all slot in slots do
5.    Marked (slot) ← false
6.  endfor
7.  while true do
8.    WaitForEvent ()
9.    if Event (ArrivalNotification) then
10.     Receive (frame)
11.     if corrupted (frame) and not naksent then
12.       SentNAk (Rn)
13.       naksent ← true
14.       Sleep()
```

Teacher's Signature:

Experiment Name / No.:

Camlin | Page No.
Date / /

```
        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
              DeliverData (Rn)
              Purge (Rn)
              Rn ← (Rn+1) % 2^m
              ackNeeded ← True
            endwhile
            if ackNeeded then
              SendAck (Rn)
              ackNeeded ← False
              nakSent ← False
            endif
          endif
        endif
endwhile
```

RESULT:

Experiment Name / No.: Link state Routing

Camlin | Page No.
Date / /

AIM: To Implement and Simulate link state protocol

Algorithm:

D(v): cost of the least cost path from the source node to the destination node v as of this Iteration of the algorithm.

P(v): previous node of v along the current least cost path from the source to v.

N': Subset of nodes. v is in N' if the least cost path from the source to v is definitely known

Initialization:

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)

Experiment Name / No.: Concurrent FTP

Camlin | Page No.
Date / /

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's 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 connection 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. ie) when file pointer reaches end of file, file transfer "completed" message is sent by the Server to the accepted client connection using newsd, Socket file descriptor.

Teacher's Signature:

Experiment Name / No.:

Camlin | Page No.
Date / /

Algorithm - Client

1. Create a 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. 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 & exit.
10. close the socket communication.

Teacher's Signature:

Experiment Name / No.: Leaky Bucket

Camlin | Page No.
Date / /

AIM: To implement Congestion Control using Leaky bucket Algorithm.

Algorithm:

1. Start
2. Set the bucket size or the buffer 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.

Experiment Name / No.: Study of Wireshark Tool

Camlin | Page No. | Date / /

AIM: To Study the Working of Wireshark tool.

Wireshark has a very rich history. Gerald combs, a compoter Science graduate of the university of missouri at Kansas city, Orginally developed it out of necessity. The first version of Combs'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 the Ethereal trademarks, and combs was unable to reach an agreement 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 protocol that it sopports more than 850 as of this writing. These range from common ones like IP and DHCP to more advanced proprietary protocols like AppleTalk and Bit Torrent.

User-friendliness: The wireshark interface is one of the easies to understand of any packet-sniffing application. It is a GUI-based with very clearly written content menus and a straightforward.

layout: It also provides several features designed to enhance usability, such as protocol-based color coding and detialed graphical representations of raw data. Unlike some of the more complicated command-line-driven alternatives, like

Teacher's Signature:

Experiment Name / No.: _____ Camlin Page No. Date / /

tcpdump, the Wireshark GUI is great for those who are just entering the world of packet analysis.
cost: Since it is open source, wireshark's pricing can't be 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 windows, Mac OSX 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:

Teacher's Signature: _____