**Week 5:Implementation of SLIDING WINDOW PROTOCOL**

**AIM:**

To write a C program to perform sliding window.

**ALGORITHM:**

1. Start the program.

2. Get the frame size from the user

3. To create the frame based on the user request.

4. To send frames to server from the client side.

5. If your frames reach the server it will send ACK signal to client otherwise it will

send NACK signal to client.

6. Stop the program

PROGRAM :

// SLIDING WINDOW PROTOCOL

Client :

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

struct mymsgbuf

{

long mtype;

char mtext[25];

};

FILE \*fp;

int main()

{

struct mymsgbuf buf;

int msgid;

int i=0,s;

int count=0,frmsz;

int a[100];

char d;

if((msgid=msgget(89,IPC\_CREAT|0666))==-1)

{

printf("\n ERROR IN MSGGET");

exit(0);

}

printf("\n Enter the frame size:");

scanf("%d",&frmsz);

if((fp=fopen("check","r"))==NULL)

printf("\n FILE NOT OPENED");

else

printf("\n FILE OPENED");

while(!feof(fp))

{

d=getc(fp);

a[i]=d;

i++;

}

s=i;

for(i=0;i<frmsz;i++) //print from the check file

printf("\t %c",a[i]);

for(i=0;i<frmsz;i++)

{ if((msgrcv(msgid,&buf,sizeof(buf),0,1))==-1)

{

printf("\n ERROR IN MSGRCV");

exit(0);

}

printf("\n RECEIVED FRAMES ARE:%c",buf.mtext[i]);

}

for(i=0;i<frmsz;i++)

{ if(a[i]==buf.mtext[i])

count++;

} if(count==0)

{

printf("\n FRAMES WERE NOT RECEIVED IN CORRECT SEQ");

exit(0);

} if(count==frmsz)

{

printf("\n FRAMES WERE RECEIVED IN CORRECT SEQ");

} else

{

printf("\n FRAMES WERE NOT RECEIVED IN CORRECT SEQ");

}}

**Sliding Window Protocol -**

**Server**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

struct mymsgbuf

{ long mtype;

char mtext[25];

};

FILE \*fp;

int main()

{s

truct mymsgbuf buf;

int si,ei,sz;

int msgid;

int i=0,s;

int a[100];

char d;

if((fp=fopen("send","r"))==NULL)

printf("\n FILE NOT OPENED");

else

printf("\n FILE OPENED");

printf("\n Enter starting and ending index of frame array:");

scanf("%d%d",&si,&ei);

sz=ei-si;

if((msgid=msgget(89,IPC\_CREAT|0666))==-1)

{

printf("\n ERROR IN MSGGET");

exit(0);

}

while(!feof(fp))

{

d=getc(fp);

a[i]=d;

i++;

}s

=i;

buf.mtype=1;

for(i=si;i<=ei;i++)

{

buf.mtext[i]=a[i];

}

for(i=si;i<=ei;i++) //the frames to be sent

printf("\t %c",buf.mtext[i]);

for(i=0;i<=sz;i++)

{ if((msgsnd(msgid,&buf,sizeof(buf),0))==-1)

{

printf("\n ERROR IN MSGSND");

exit(0);

}}

printf("\n FRAMES SENT");

return 0;

}

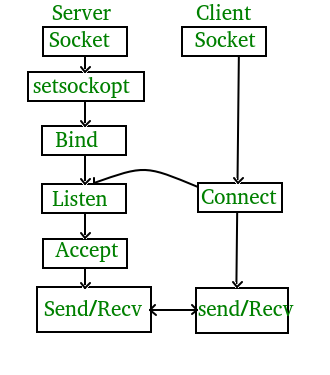
Week 6:Connection Oriented Server-Client application with TCP

[Server-Client implementation in C](https://www.geeksforgeeks.org/udp-server-client-implementation-c/)  
If we are creating a connection between client and server using TCP then it has few functionality like, TCP is suited for applications that require high reliability, and transmission time is relatively less critical. It is used by other protocols like HTTP, HTTPs, FTP, SMTP, Telnet. TCP rearranges data packets in the order specified. There is absolute guarantee that the data transferred remains intact and arrives in the same order in which it was sent. TCP does Flow Control and requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control. It also does error checking and error recovery. Erroneous packets are retransmitted from the source to the destination.

The entire process can be broken down into following steps:

**TCP Server –**

**TCP Server –**

**TCP Server –**

1. using create(), Create TCP socket.
2. using bind(), Bind the socket to server address.
3. using listen(), put the server socket in a passive mode, where it waits for the client to approach the server to make a connection
4. using accept(), At this point, connection is established between client and server, and they are ready to transfer data.
5. Go back to Step 3.

**TCP Client –**

1. Create TCP socket.
2. connect newly created client socket to server.

**TCP Server:**

|  |
| --- |
| #include <stdio.h>  #include <netdb.h>  #include <netinet/in.h>  #include <stdlib.h>  #include <string.h>  #include <sys/socket.h>  #include <sys/types.h>  #define MAX 80  #define PORT 8080  #define SA struct sockaddr    // Function designed for chat between client and server.  void func(int sockfd)  {      char buff[MAX];      int n;      // infinite loop for chat      for (;;) {          bzero(buff, MAX);            // read the message from client and copy it in buffer          read(sockfd, buff, sizeof(buff));          // print buffer which contains the client contents          printf("From client: %s\t To client : ", buff);          bzero(buff, MAX);          n = 0;          // copy server message in the buffer          while ((buff[n++] = getchar()) != '\n')              ;            // and send that buffer to client          write(sockfd, buff, sizeof(buff));            // if msg contains "Exit" then server exit and chat ended.          if (strncmp("exit", buff, 4) == 0) {              printf("Server Exit...\n");              break;          }      }  }    // Driver function  int main()  {      int sockfd, connfd, len;      struct sockaddr\_in servaddr, cli;        // socket create and verification      sockfd = socket(AF\_INET, SOCK\_STREAM, 0);      if (sockfd == -1) {          printf("socket creation failed...\n");          exit(0);      }      else          printf("Socket successfully created..\n");      bzero(&servaddr, sizeof(servaddr));        // assign IP, PORT      servaddr.sin\_family = AF\_INET;      servaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);      servaddr.sin\_port = htons(PORT);        // Binding newly created socket to given IP and verification      if ((bind(sockfd, (SA\*)&servaddr, sizeof(servaddr))) != 0) {          printf("socket bind failed...\n");          exit(0);      }      else          printf("Socket successfully binded..\n");        // Now server is ready to listen and verification      if ((listen(sockfd, 5)) != 0) {          printf("Listen failed...\n");          exit(0);      }      else          printf("Server listening..\n");      len = sizeof(cli);        // Accept the data packet from client and verification      connfd = accept(sockfd, (SA\*)&cli, &len);      if (connfd < 0) {          printf("server acccept failed...\n");          exit(0);      }      else          printf("server acccept the client...\n");        // Function for chatting between client and server      func(connfd);        // After chatting close the socket      close(sockfd);  } |

**TCP Client:**

|  |
| --- |
| #include <netdb.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <sys/socket.h>  #define MAX 80  #define PORT 8080  #define SA struct sockaddr  void func(int sockfd)  {      char buff[MAX];      int n;      for (;;) {          bzero(buff, sizeof(buff));          printf("Enter the string : ");          n = 0;          while ((buff[n++] = getchar()) != '\n')              ;          write(sockfd, buff, sizeof(buff));          bzero(buff, sizeof(buff));          read(sockfd, buff, sizeof(buff));          printf("From Server : %s", buff);          if ((strncmp(buff, "exit", 4)) == 0) {              printf("Client Exit...\n");              break;          }      }  }    int main()  {      int sockfd, connfd;      struct sockaddr\_in servaddr, cli;        // socket create and varification      sockfd = socket(AF\_INET, SOCK\_STREAM, 0);      if (sockfd == -1) {          printf("socket creation failed...\n");          exit(0);      }      else          printf("Socket successfully created..\n");      bzero(&servaddr, sizeof(servaddr));        // assign IP, PORT      servaddr.sin\_family = AF\_INET;      servaddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");      servaddr.sin\_port = htons(PORT);        // connect the client socket to server socket      if (connect(sockfd, (SA\*)&servaddr, sizeof(servaddr)) != 0) {          printf("connection with the server failed...\n");          exit(0);      }      else          printf("connected to the server..\n");        // function for chat      func(sockfd);        // close the socket      close(sockfd);  } |

**Compilation –**  
Server side:  
gcc server.c -o server  
./server

Client side:  
gcc client.c -o client  
./client

**Output –**  
Server side:

Socket successfully created..

Socket successfully binded..

Server listening..

server acccept the client...

From client: hi

To client : hello

From client: exit

To client : exit

Server Exit...

Client side:

Socket successfully created..

connected to the server..

Enter the string : hi

From Server : hello

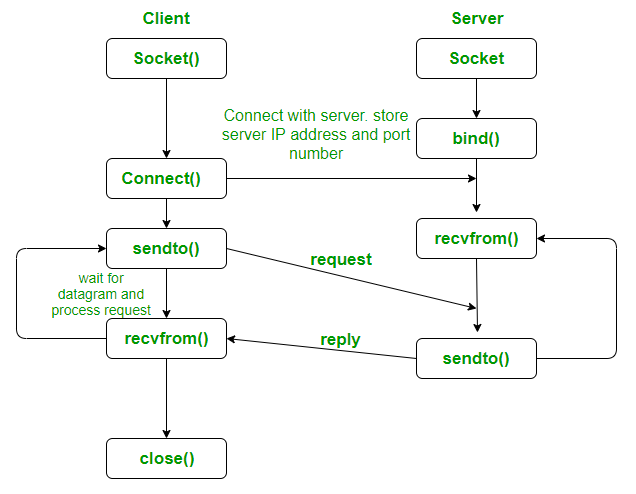
Enter the string : exit

From Server : exit

Client Exit...

Week 7 Connection Oriented Server-Client application with UDP

In UDP, the client does not form a connection with the server like in TCP and instead, It just sends a datagram. Similarly, the server need not to accept a connection and just waits for datagrams to arrive. We can call a function called **connect()** in UDP but it does not result anything like it does in TCP. There is no 3 way handshake. It just checks for any immediate errors and store the peer’s IP address and port number. connect() is storing peers address so no need to pass **server address** and **server address length** arguments in **sendto()**.



**UDP Server :**

1. Create UDP socket.
2. Bind the socket to server address.
3. Wait until datagram packet arrives from client.
4. Process the datagram packet and send a reply to client.
5. Go back to Step 3.

**UDP Client :**

1. Create UDP socket.
2. Send message to server.
3. Wait until response from server is recieved.
4. Process reply and go back to step 2, if necessary.
5. Close socket descriptor and exit.

**UDPServer.c**

|  |
| --- |
| 1. #include <stdio.h> 2. #include <stdlib.h> 3. #include <unistd.h> 4. #include <string.h> 5. #include <sys/types.h> 6. #include <sys/socket.h> 7. #include <arpa/inet.h> 8. #include <netinet/in.h> 9. #define PORT     8080 10. #define MAXLINE 1024 12. // Driver code 13. int main() { 14. int sockfd; 15. char buffer[MAXLINE]; 16. char \*hello = "Hello from server"; 17. struct sockaddr\_in servaddr, cliaddr; 19. // Creating socket file descriptor 20. if ( (sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0 ) { 21. perror("socket creation failed"); 22. exit(EXIT\_FAILURE); 23. } 25. memset(&servaddr, 0, sizeof(servaddr)); 26. memset(&cliaddr, 0, sizeof(cliaddr)); 28. // Filling server information 29. servaddr.sin\_family    = AF\_INET; // IPv4 30. servaddr.sin\_addr.s\_addr = INADDR\_ANY; 31. servaddr.sin\_port = htons(PORT); 33. // Bind the socket with the server address 34. if ( bind(sockfd, (const struct sockaddr \*)&servaddr, 35. sizeof(servaddr)) < 0 ) 36. { 37. perror("bind failed"); 38. exit(EXIT\_FAILURE); 39. } 41. int len, n; 43. len = sizeof(cliaddr);  //len is value/resuslt 45. n = recvfrom(sockfd, (char \*)buffer, MAXLINE, 46. MSG\_WAITALL, ( struct sockaddr \*) &cliaddr, 47. &len); 48. buffer[n] = '\0'; 49. printf("Client : %s\n", buffer); 50. sendto(sockfd, (const char \*)hello, strlen(hello), 51. MSG\_CONFIRM, (const struct sockaddr \*) &cliaddr, 52. len); 53. printf("Hello message sent.\n"); 55. return 0; 56. } |

**UDPClient.c**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <string.h>  #include <sys/types.h>  #include <sys/socket.h>  #include <arpa/inet.h>  #include <netinet/in.h>    #define PORT     8080  #define MAXLINE 1024    // Driver code  int main() {      int sockfd;      char buffer[MAXLINE];      char \*hello = "Hello from client";      struct sockaddr\_in     servaddr;        // Creating socket file descriptor      if ( (sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0 ) {          perror("socket creation failed");          exit(EXIT\_FAILURE);      }        memset(&servaddr, 0, sizeof(servaddr));        // Filling server information      servaddr.sin\_family = AF\_INET;      servaddr.sin\_port = htons(PORT);      servaddr.sin\_addr.s\_addr = INADDR\_ANY;        int n, len;        sendto(sockfd, (const char \*)hello, strlen(hello),          MSG\_CONFIRM, (const struct sockaddr \*) &servaddr,              sizeof(servaddr));      printf("Hello message sent.\n");        n = recvfrom(sockfd, (char \*)buffer, MAXLINE,                  MSG\_WAITALL, (struct sockaddr \*) &servaddr,                  &len);      buffer[n] = '\0';      printf("Server : %s\n", buffer);        close(sockfd);      return 0;  } |

**Output :**

$ ./server

Client : Hello from client

Hello message sent.

$ ./client

Hello message sent.

Server : Hello from server