EXPERIMENT 7

AIM:

To implement and simulate algorithm for Simple Mail Transfer Protocol.

THEORY:

SMTP is part of the application layer of the TCP/IP protocol and traditionally operates on port 25. It utilizes a process called “store and forward” which is used to orchestrate sending your email across different networks. Within the SMTP protocol, there are smaller software services called Mail Transfer Agents that help manage the transfer of the email and its final delivery to a recipient’s mailbox. Not only does SMTP define this entire communication flow, it can also support delayed delivery of an email either at the sender site, receiver site, or at any intermediate server.

ALGORITHM:

Server:-

1. Create a socket with type as SOCK_DGRAM to create a udp socket using socket() system call.Create a socket with type as SOCK_DGRAM to create a udp socket using socket() system call.
2. Bind the socket to a specific port using bind() system call.
3. Receive first packet from client using recvfrom() and reply with status code 220 with fully qualified domain name to indicate server ready.
4. Receive HELO command with fully qualified domain name from client using recvfrom() system call and respond with status code 250.
5. Receive MAIL command with FROM address from client using recvfrom() system call and respond with status code 250.
6. Receive RCPT command with TO address from client using recvfrom() system call and respond with status code 250.
7. Receive DATA command from client using recvfrom() system call and respond with status code 354 to start receiving mail body.
8. Receive the mail body from the client using recvfrom() system call.
9. Receive QUIT command] from client using recvfrom() system call and respond with status code 221.
10. Close the socket.

Client:-

1. Create a socket with type as SOCK_DGRAM to create a udp socket using socket() system call.
2. Send an initial message to the server using sendto() and expect a response with status code 220 and fully qualified domain name of the server.
3. Send a HELO command to the server using sendto() system call with a fully qualified domain name and expect a response with status code 250.

4. Send a MAIL command along with FROM address to the server using sendto() system call and expect a response with status code 250.
5. Send a RCPT command along with TO address to the server using sendto() system call and expect a response with status code 250.
6. Send a DATA command to the server using sendto() system call and expect a response with status code 354.
7. Read the body of the mail from the user and send it to the server using sendto() system call.
8. Send a QUIT command to the server using sendto() system call and expect a response with status code 221.
9. Close the socket

PROGRAM

Server:

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<stdlib.h>
#include<sys/types.h>
#include<netinet/in.h>
#define BUF_SIZE 256
int main(int argc,char* argv[])
{
    struct sockaddr_in server,client;
    char str[50],msg[20];
    if(argc!=2)
    printf("Input format not correct");
    int sockfd=socket(AF_INET,SOCK_DGRAM,0);
    if(sockfd==-1)
    printf("Error in socket();");
    server.sin_family=AF_INET;
    server.sin_addr.s_addr=INADDR_ANY;
    server.sin_port=htons(atoi(argv[1]));
    client.sin_family=AF_INET;
    client.sin_addr.s_addr=INADDR_ANY;
    client.sin_port=htons(atoi(argv[1]));
    if(bind(sockfd,(struct sockaddr *)&server,sizeof(server))<0)
    printf("Error in bind()! \n");
    socklen_t client_len=sizeof(client);
    printf("server waiting......");
    sleep(3);
```

```
if(recvfrom(sockfd,str,100,0,(struct sockaddr *)&client,&client_len)<0)
printf("Error in recvfrom()!");
printf("\nGot message from client:%s",str);
printf("\nSending greeting message to client");
strcpy(str,"220 127.0.0.1");
sleep(10);
if(sendto(sockfd,str,sizeof(str),0,(struct sockaddr *)&client,sizeof(client))<0)
printf("Error in send");
sleep(3);
if((recvfrom(sockfd,str,sizeof(str),0,(struct sockaddr *)&client,&client_len)),0)
printf("Error in recv");
if(strncmp(str,"HELO",4))
printf("\n'HELO' expected from client....");
printf("\n%s",str);
printf("\nSending response...");
strcpy(str,"250 ok");
if(sendto(sockfd,str,sizeof(str),0,(struct sockaddr *)&client,sizeof(client))<0)
printf("Error in send");
sleep(3);
if((recvfrom(sockfd,str,sizeof(str),0,(struct sockaddr *)&client,&client_len))<0)
printf("Error in recv");
if(strncmp(str,"MAIL FROM",9))
printf("MAIL FROM expected from client...");
printf("\n%s",str);
printf("\nSending response....");
strcpy(str,"250 ok");
if(sendto(sockfd,str,sizeof(str),0,(struct sockaddr *)&client,sizeof(client))<0)
printf("Error in send");
sleep(3);
if((recvfrom(sockfd,str,sizeof(str),0,(struct sockaddr *)&client,&client_len))<0)
printf("Error in recv");
if(strncmp(str,"RCPT TO",7))
printf("\nRCPT TO expected from client....");
printf("\n%s",str);
printf("\nSending response....");
strcpy(str,"250 ok");
if(sendto(sockfd,str,sizeof(str),0,(struct sockaddr *)&client,sizeof(client))<0)
printf("Error in send");
sleep(3);
if((recvfrom(sockfd,str,sizeof(str),0,(struct sockaddr *)&client,&client_len))<0)
printf("Error in recv");
if(strncmp(str,"DATA",4))
printf("\nDATA expected from client....");
printf("\n%s",str);
printf("\nSending response....");
```

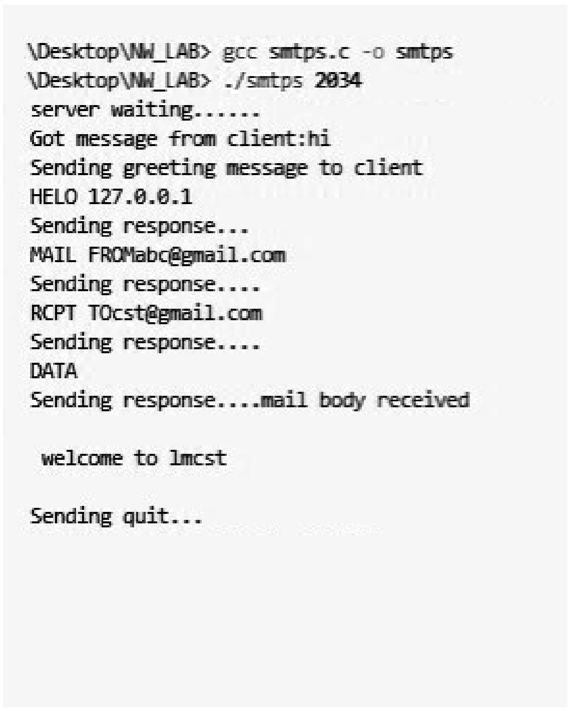
```
    strcpy(str,"354 Go ahead");
    if(sendto(sockfd,str,sizeof(str),0,(struct sockaddr *)&client,sizeof(client))<0)
    printf("Error in send");
    if((recvfrom(sockfd,msg,sizeof(str),0,(struct sockaddr *)&client,&client_len))<0)
    printf("Error in recv");
    printf("mail body received");
    printf("\n%s",msg);
    if((recvfrom(sockfd,str,sizeof(str),0,(struct sockaddr *)&client,&client_len))<0)
    printf("Error in recv");
    if(strncmp(str,"QUIT",4))
    printf("quit expected from client....");
    printf("\nSending quit...");
    strcpy(str,"221 OK");
    if(sendto(sockfd,str,sizeof(str),0,(struct sockaddr *)&client,sizeof(client))<0)
    printf("Error in send");
    close(sockfd);
    return 0;
}
```

Client:

```
#include<string.h>
#include<sys/socket.h>
#include<netdb.h>
#include<stdlib.h>
#include<stdio.h>
#include<netinet/in.h>
#include<sys/types.h>
#include<netinet/in.h>
#define BUF_SIZE 256
int main(int argc,char* argv[])
{
    struct sockaddr_in server,client;
    char str[50]="hi";
    char mail_f[50],mail_to[50],msg[20],c;
    int t=0;
    socklen_t l=sizeof(server);
    if(argc!=3)
    printf("Input format not correct");
    int sockfd=socket(AF_INET,SOCK_DGRAM,0);
    if(sockfd==-1)
    printf("Error in socket();");
    server.sin_family=AF_INET;
    server.sin_addr.s_addr=INADDR_ANY;
    server.sin_port=htons(atoi(argv[2]));
    client.sin_family=AF_INET;
client.sin_addr.s_addr=INADDR_ANY;
client.sin_port=htons(atoi(argv[2]));
printf("Sending hi to server");
sleep(10);
if(sendto(sockfd,str,sizeof(str),0,(struct sockaddr*)&server,sizeof(server))<0)
printf("Error in sento");
if(recvfrom(sockfd,str,sizeof(str),0,(struct sockaddr *)&server,&l)<0)
printf("Error in recv");
printf("\ngreeting msg is %s",str);
if(strncmp(str,"220",3))
printf("\nConn not established \n code 220 expected");
printf("\nSending HELO");
strcpy(str,"HELO 127.0.0.1");
if(sendto(sockfd,str,sizeof(str),0,(struct sockaddr *)&server,sizeof(server))<0)
printf("Error in sendto");
sleep(3);
printf("\nReceiving from server");
if((recvfrom(sockfd,str,sizeof(str),0,(struct sockaddr *)&server,&l))<0)
printf("Error in recv");
if(strncmp(str,"250",3))
printf("\nOk not received from server");
printf("\nServer has send %s",str);
printf("\nEnter FROM address\n");
scanf("%s",mail_f);
strcpy(str,"MAIL FROM");
strcat(str,mail_f);
if(sendto(sockfd,str,sizeof(str),0,(struct sockaddr *)&server,sizeof(server))<0)
printf("Error in sendto");
sleep(3);
if((recvfrom(sockfd,str,sizeof(str),0,(struct sockaddr *)&server,&l))<0)
printf("Error in recv");
if(strncmp(str,"250",3))
printf("\nOk not received from server");
printf("%s",str);
printf("\nEnter TO address\n");
scanf("%s",mail_to);
strcpy(str,"RCPT TO");
strcat(str,mail_to);
if(sendto(sockfd,str,sizeof(str),0,(struct sockaddr *)&server,sizeof(server))<0)
printf("Error in sendto");
sleep(3);
if((recvfrom(sockfd,str,sizeof(str),0,(struct sockaddr *)&server,&l))<0)
printf("Error in recv");
if(strncmp(str,"250",3))
printf("\nOk not received from server");
```

```
    printf("%s",str);
    printf("\nSending DATA to server");
    strcpy(str,"DATA");
    if(sendto(sockfd,str,sizeof(str),0,(struct sockaddr *)&server,sizeof(server))<0)
    printf("Error in sendto");
    sleep(3);
    if((recvfrom(sockfd,str,sizeof(str),0,(struct sockaddr *)&server,&l))<0)
    printf("Error in recv");
    if(strncmp(str,"354",3))
    printf("\nOk not received from server");
    printf("%s",str);
    printf("\nEnter mail body");
    while(1)
    {
        c=getchar();
        if(c=='$')
        {
            msg[t]='\0';
            break;
        }
        if(c=='\0')
        continue;
        msg[t++]=c;
    }
    if(sendto(sockfd,msg,sizeof(msg),0,(struct sockaddr *)&server,sizeof(server))<0)
    printf("Error in sendto");
    sleep(3);
    printf("\nSending QUIT to server");
    strcpy(str,"QUIT");
    if(sendto(sockfd,str,sizeof(str),0,(struct sockaddr *)&server,sizeof(server))<0)
    printf("Error in sendto");
    if((recvfrom(sockfd,str,sizeof(str),0,(struct sockaddr *)&server,&l))<0)
    printf("Error in recv");
    if(strncmp(str,"221",3))
    printf("\nOk not received from server");
    printf("\nServer has send GOODBYE.....Closing conn\n");
    printf("\n Bye");
    close(sockfd);
    return 0;
}
```

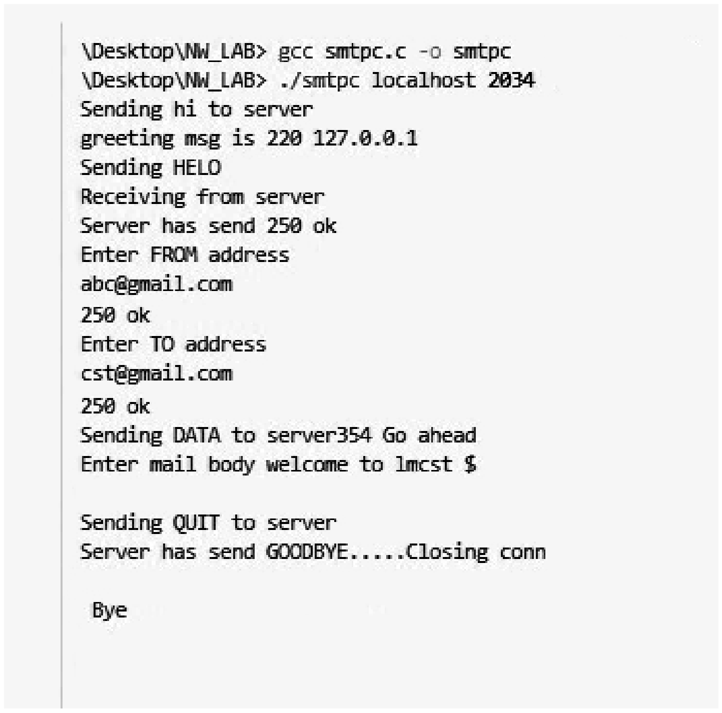
OUTPUT



```
\Desktop\NW_LAB> gcc smtps.c -o smtps
\Desktop\NW_LAB> ./smtps 2034
server waiting......
Got message from client:hi
Sending greeting message to client
HELO 127.0.0.1
Sending response...
MAIL FROMabc@gmail.com
Sending response....
RCPT TOcst@gmail.com
Sending response....
DATA
Sending response....mail body received

 welcome to lmcst

Sending quit...
```



```
\Desktop\NW_LAB> gcc smtpc.c -o smtpc
\Desktop\NW_LAB> ./smtpc localhost 2034
Sending hi to server
greeting msg is 220 127.0.0.1
Sending HELO
Receiving from server
Server has send 250 ok
Enter FROM address
abc@gmail.com
250 ok
Enter TO address
cst@gmail.com
250 ok
Sending DATA to server354 Go ahead
Enter mail body welcome to lmcst $

Sending QUIT to server
Server has send GOODBYE.....Closing conn

 Bye
```

RESULT:

The experiment was executed successfully.

EXPERIMENT 8

AIM:

To implement and simulate algorithm for File Transfer Protocol.

THEORY:

FTP, which stands for File Transfer Protocol, was developed in the 1970s to allow files to be transferred between a client and a server on a computer network. The FTP protocol uses two separate channels — the command (or control) channel and the data channel — to exchange files. The command channel is responsible for accepting client connections and executing other simple commands. It typically uses server port 21. FTP clients will connect to this port to initiate a conversation for file transfer and authenticate themselves by sending a username and password.

After authentication, the client and server will then negotiate a new common server port for the data channel, over which the file will be transferred. Once the file transfer is complete, the data channel is closed. If multiple files are to be sent concurrently, a range of data channel ports must be used. The control channel remains idle until the file transfer is complete. It then reports that the file transfer was either successful or failed.

ALGORITHM:

Server:-

1. Create socket using socket() system call with address family AF_INET, type SOCK_STREAM and default protocol.
2. Bind server address and port using bind() system call.
3. Whai for client connection to complete accepting connections using accept() system call.
4. Receive the client file using recv() system call.
5. Using fgets(char *str, int n, FILE *stream) function, we need a line of text from the specified stream and store it into the string pointed by str. It stops when either (n-1) characters are read or when the end of file is reached.
6. On successful execution is when the file pointer reaches the end of file, file transfer “completed” message is sent by server to accepted client connection.

Client:-

1. Create socket using socket() system call with address family AF_INET, type SOCK_STREAM and default protocol.
2. Enter the client port number.
3. Fill in the internal socket address structure
4. Connect to the server address using connect() system call.
5. Read the existing and new file name from the user.
6. Send existing file to server using send() system call.
7. Receive feedback from server “completed” regarding file transfer completion.
8. Write file is transferred to the standard output stream.
9. Close the socket connection and file pointer

PROGRAM

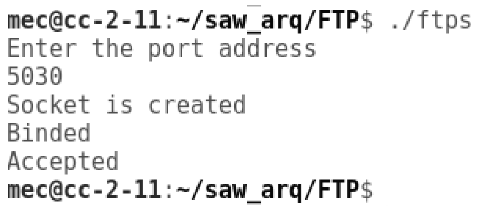
Server:

```
#include<stdio.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
FILE *fp;
int sd,newsd,ser,n,a,cli,pid,bd,port,clilen;
char name[100],fileread[100],fname[100],ch,file[100],rcv[100];
struct sockaddr_in servaddr,cliaddr;
printf("Enter the port address\n");
scanf("%d",&port);
sd=socket(AF_INET,SOCK_STREAM,0);
if(sd<0)
printf("Cant create\n");
else
printf("Socket is created\n");
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
servaddr.sin_port=htons(port);
a=sizeof(servaddr);
bd=bind(sd,(struct sockaddr *)&servaddr,a);
if(bd<0)
printf("Cant bind\n");
else
printf("Binded\n");
listen(sd,5);
clilen=sizeof(cliaddr);
newsd=accept(sd,(struct sockaddr *)&cliaddr,&clilen);
if(newsd<0)
{
printf("Cant accept\n");
}
else
printf("Accepted\n");
n=recv(newsd,rcv,100,0);
rcv[n]='\0';
fp=fopen(rcv,"r");
if(fp==NULL)
{
send(newsd,"error",5,0);
close(newsd);
}
else
{
while(fgets(fileread,sizeof(fileread),fp))
{
if(send(newsd,fileread,sizeof(fileread),0)<0)
{
printf("Can’t send file contents\n");
}
sleep(1);
}
if(!fgets(fileread,sizeof(fileread),fp))
{
//when file pointer reaches end of file, file transfer “completed” message is send to
accepted client connection using newsd, socket file descriptor.
send(newsd,"completed",999999999,0);
}
return(0);
}
}
```

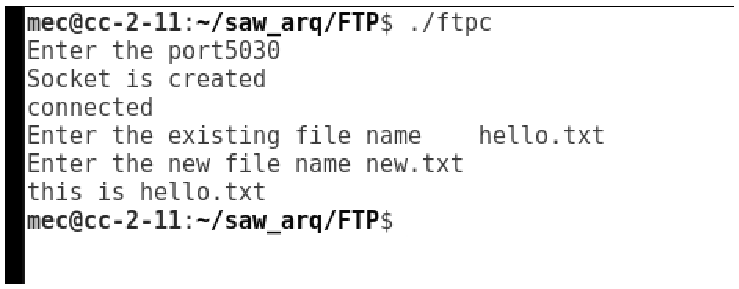
Client:

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<unistd.h>
int main()
{
FILE *fp;
int csd,n,ser,s,cli,cport,newsd;
char name[100],rcvmsg[100],rcvg[100],fname[100];
struct sockaddr_in servaddr;
printf("Enter the port");
scanf("%d",&cport);
csd=socket(AF_INET,SOCK_STREAM,0);
if(csd<0)
{
printf("Error....\n");
exit(0);
}
else
printf("Socket is created\n");
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
servaddr.sin_port=htons(cport);
if(connect(csd,(struct sockaddr *)&servaddr,sizeof(servaddr))<0)
printf("Error in connection\n");
else
printf("connected\n");
printf("Enter the existing file name\t");
scanf("%s",name);
printf("Enter the new file name\t");
scanf("%s",fname);
fp=fopen(fname,"w");
send(csd,name,sizeof(name),0);
while(1)
{
s=recv(csd,rcvg,100,0);
rcvg[s]='\0';
if(strcmp(rcvg,"error")==0)
printf("File is not available\n");
if(strcmp(rcvg,"completed")==0)
{
printf("File is transferred........\n");
fclose(fp);
close(csd);
break;
}
else
fputs(rcvg,stdout);
fprintf(fp,"%s",rcvg);
return 0;
}
}
```

OUTPUT



```
mec@cc-2-11:~/saw_arq/FTP$ ./ftps
Enter the port address
5030
Socket is created
Binded
Accepted
mec@cc-2-11:~/saw_arq/FTP$
```



```
mec@cc-2-11:~/saw_arq/FTP$ ./ftpc
Enter the port5030
Socket is created
connected
Enter the existing file name    hello.txt
Enter the new file name new.txt
this is hello.txt
mec@cc-2-11:~/saw_arq/FTP$
```

RESULT:The experiment was executed successfully.

EXPERIMENT 9

AIM:

To implement congestion control using leaky bucket algorithm

THEORY:

The Leaky Bucket Algorithm used to control rate in a network. It is implemented as a single-server queue with constant service time. If the bucket (buffer) overflows then packets are discarded. It enforces a constant output rate (average rate) regardless of the burstiness of the input. It does nothing when input is idle. The host injects one packet per clock tick onto the network. This results in a uniform flow of packets, smoothing out bursts and reducing congestion.When packets are the same size (as in ATM cells), the one packet per tick is okay. For variable length packets though, it is better to allow a fixed number of bytes per tick.

ALGORITHM:

1. Start
2. Set bucket size or buffer size.
3. Set output rate.
4. Transmit the packets such that there is no overflow.
5. Repeat the process of transmission until all packets are transmitted. (Reject packets where its size is greater than the bucket size).
6. Stop

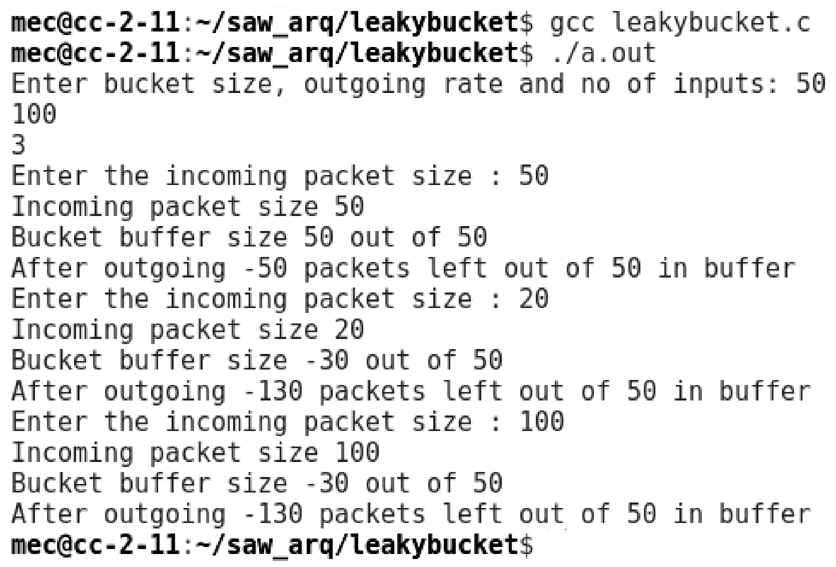
PROGRAM

```
#include<stdio.h>
int main(){
   int incoming, outgoing, buck_size, n, store = 0;
   printf("Enter bucket size, outgoing rate and no of inputs: ");
   scanf("%d %d %d", &buck_size, &outgoing, &n);

   while (n != 0) {
      printf("Enter the incoming packet size : ");
      scanf("%d", &incoming);
      printf("Incoming packet size %d\n", incoming);
      if (incoming <= (buck_size - store)){
         store += incoming;
         printf("Bucket buffer size %d out of %d\n", store, buck_size);
      } else {
         printf("Dropped %d no of packets\n", incoming - (buck_size - store));
         printf("Bucket buffer size %d out of %d\n", store, buck_size);
         store = buck_size;
      }
```

```
        store = store - outgoing;
        printf("After outgoing %d packets left out of %d in buffer\n", store, buck_size);
        n--;
    }
}
```

OUTPUT



```
mec@cc-2-11:~/saw_arq/leakybucket$ gcc leakybucket.c
mec@cc-2-11:~/saw_arq/leakybucket$ ./a.out
Enter bucket size, outgoing rate and no of inputs: 50
100
3
Enter the incoming packet size : 50
Incoming packet size 50
Bucket buffer size 50 out of 50
After outgoing -50 packets left out of 50 in buffer
Enter the incoming packet size : 20
Incoming packet size 20
Bucket buffer size -30 out of 50
After outgoing -130 packets left out of 50 in buffer
Enter the incoming packet size : 100
Incoming packet size 100
Bucket buffer size -30 out of 50
After outgoing -130 packets left out of 50 in buffer
mec@cc-2-11:~/saw_arq/leakybucket$
```

RESULT:

The experiment was executed successfully.