Read Me
Assignment3-Ques2

Part 0:

Random String generator

```c
char *randstring(size_t length,char *String,int len) {

    char *result=malloc(sizeof(char)*(length+1));
    if(result!=NULL){
        for(int i=0;i<length;i++){
            int k=rand()%len;
            result[i]=String[k];
            sleep(0.1);
        }
        result[length]='\0';
    }
    return result;
}
```

Function to generate a random string of length 20 terminated by null character. A sleep(0.1) seconds to ensure strings are different every time we run.

Part 1

Unix-Domain Sockets

Setting up of the socket in the sender.(Server side).

```c
struct sockaddr_un addr;
int sd,cd;
if(remove(SV_SOCK_PATH)==-1 && errno!=ENOENT){
    perror("Socket removed");
}
sd=socket(AF_UNIX,SOCK_STREAM,0);
if(sd==-1){
    perror("Socket cant be created");
}
memset(&addr,0,sizeof(struct sockaddr_un));
addr.sun_family=AF_UNIX;
strncpy(addr.sun_path,SV_SOCK_PATH,sizeof(struct sockaddr_un)-1);
bind(sd,(struct sockaddr *)&addr,sizeof(struct sockaddr_un));
listen(sd,2);
cd=accept(sd,NULL,NULL);
```

```
for(int i=0;i<50;i++){
    int j=0;
    while(j<5){
        int x=i+j;
        send(cd,array[x],sizeof(char)*21,0);
        send(cd,&x,sizeof(int),0);
        j++;
    }
    int *k=malloc(sizeof(int));
    recv(cd,k,sizeof(int),0);
    printf("%d\n",*k);
    i=(*k);
```

The code above is for the sender while the one below is for receiver.The string and integer are sent separately and received separately.

```
char buffer[21];
int n;
for(int i=0;i<50;i++){
    int j=0;
    while(j<5){
        recv(sf,buffer,sizeof(char)*21,0);
        recv(sf,&n,sizeof(int),0);
        printf("%s %d\n",buffer,n);
        j++;
    }
    int d=n;
    printf("%d\n",d);
    send(sf,&d,sizeof(int),0);
    i=d;
```

Output:

The output on the sender and receiver is given.

```
[artix Assignment3_Ques2]# ./socket1
4
9
14
19
24
29
34
39
44
49

Total time taken 0.000954 seconds
```

```
rUSFzKXSFxVxycEQZtuB 22
iJLBVqxoYQdpKyUjiRDp 23
ozModqeclyguhRVFJUvH 24
24
mzXWzRIHllzbmlqpeuUp 25
TalCRIhcDfJPgGoFayOl 26
lnmybCpfXjWsjKwCsgEV 27
lQnrwdWYBmjMAwMdAciZ 28
NGrzqODlwKgHAVYyYwxb 29
29
IGqKEcNfGVeueYvWoYhK 30
IQriLRjmnGnxmDitiVyq 31
temZEHVSideQvXBgOKUD 32
SibGnLZXGZoZGcYkKwdu 33
bhMwfnevzZBTJCAynzYv 34
34
bOXHQxRcTwwwgjSlYzia 35
aJTlNtJCvjYYXVFOUwQq 36
vPMBYhoWGwyiIstvnEYI 37
OyGLVNzPMSfhHSIhZWFh 38
UepCwIYJOyUCWcQRQpiC 39
39
JQJSITZjPFSJlhmHrMSf 40
kMIIrazhpjlBZUtjpVse 41
ckpnTdwkPoqBDAjUAkDr 42
uOutkqDzlxfPkWcFaaQr 43
oITtKcPMnsFHIaCuqHTd 44
44
HAStxwYZwqqmYjikmXWb 45
SCkAeoxWWqBftVyQrwPp 46
pHdnRlyFLWiDasFeGcae 47
VdjoYHGpGXgVEkKXXKCI 48
jmNjEVolXQRuUbkukQms 49
49
```

Part 2

Shared Memory

Setting up of shared memory segment.2 shared memory segments are used,1 for the sending of the strings and one for receiving the acknowledgements. The strings are sent along with their IDs concatenated.

```c
    printf("%s\n",array[i]);
}
int SIZE=512;
int sh;
if((sh=shm_open("p",O_RDWR|O_CREAT,0777))==-1){
    perror("Error creating shm");
}
ftruncate(sh,SIZE);
void *p=mmap(NULL,SIZE,PROT_WRITE,MAP_SHARED,sh,0);
if(p<0){
    perror("Mapping error");
}
int sh2;
if((sh2=shm_open("pp",O_RDWR|O_CREAT,0777))==-1){
    perror("Error creating shm");
```

```
for(int i=0;i<10;i++){
    for(int j=0;j<5;j++){
        char *k=array[i*5+j];
        char st[4];
        sprintf(st,"%d",(i*5+j));
        strcat(k," ");
        strcat(k,st);
        memcpy(p,k,sizeof(char)*24);
        sem_post(sem_des);
        sleep(0.25);
        sem_wait(semdes);
        if(fstat(sh2,&rr)==-1){
            perror("Read error");
        }
        int *p2=(int *)mmap(NULL,256,PROT_READ,MAP_SHARED,sh2,0);
        if(p2==MAP_FAILED){
            perror("Mapping error");
        }
```

To synchronize the two processes the named semaphore is used.It ensures that only in case the acknowledgement is received the sender sends data further.

```
for(int i=0;i<10;i++){
    for(int j=0;j<5;j++){
        sem_wait(sem_des);
        if(fstat(sh,&rr)==-1){
            perror("Read error");
        }
        char *p2=mmap(NULL,256,PROT_READ,MAP_SHARED,sh2,0);
        if(p2==MAP_FAILED){
            perror("Mapping error");
        }
        write(STDOUT_FILENO,p2,rr.st_size);
        //printf(" %d",(i*5+j));
        printf("\n");
        sem_post(sem_des);
        sleep(0.25);
        int x=i*5+j;
```

Output:

```
4
9
14
19
24
29
34
39
44
49
 Total time taken0.352236 seconds
```

```
e<gu,M_RI)KVnfdqZzkt 22
<AzrcwlGiO2pK8AIc0rK 23
m#vz=1JORT=Nmo&rA'!L 24
Li@nJtVMJ/oVX9cM@LpA 25
xfcCw+T!)83AJmOi51X; 26
w-Ub3!Nm0(PQ<UhsI$a7 27
u4-$GE<>_Q<nKOo:3#hW 28
QW2J5C&:y*3Iot3X!xRQ 29
:L$Ir'DN#Ny71T,Zo*VC 30
Pgc(25MkVs!R(JrfJVS0 31
0.!SS@6+Lj0tp2hK@TUB 32
>Dh$e2+NcJ:5IYctH9hk 33
LB3@(B3MmddgGn=NFYpK 34
wn:_-agACqKdR;&DAmhM 35
snT@1I3+yLCNYAARAJg/ 36
!R:9N#>c;cRppz,J=2Bq 37
:/&U2w0#+AXRjSfW9a@9 38
f7Ox!P)P.x(:sSPN;'4* 39
AhBJ#KuuKdTPBz?q<@_< 40
gW+16G<3Gyq;8VFpu$C) 41
(kEz98I3s6zyhnQEWM=r 42
=in0$TJfBa?zl8PN)m:e 43
Iu5TAdplQh3IpJ?,rAgV 44
DbaR9Suy&p55D@g:)vhF 45
2DdHb'Iw7P./QcKhU&6H 46
b@?HK'DAvN>Qfamj'Y87 47
580dA=knTJUX3Lu/LX5+ 48
z0E73QJ16'Y@8g&-:'dR 49
```

Part3
Fifo

Sender code to create fifo called testfifo and then open it in WR_ONLY to write onto fifo. Then the fifo is closed and opened in RD_ONLY to read acknowledgements. The API of fifo itself ensures synchronization.

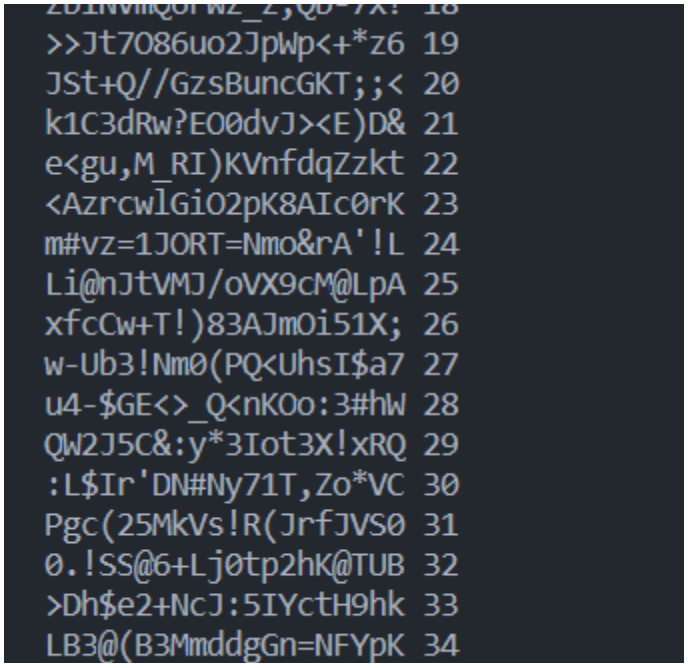
```
        perror("Some error opening to write");
    }
    char *x=malloc(sizeof(char)*21);
    for(int j=0;j<5;j++){
        x=array[i];
        if(write(fd,array[i*5+j],sizeof(char)*21)==-1){
            perror("Some error writing");
        }
        int x=i*5+j;
        if(write(fd,&x,sizeof(int))==-1){
            perror("Some error writing");
        }
    }
    close(fd);
    fd=open("testfifo",O_RDONLY);
    if(fd==-1){
        perror("Some error");
    }
```

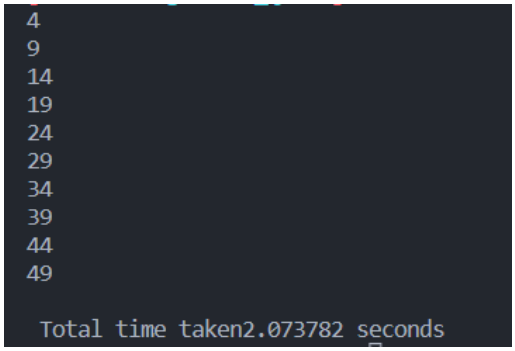
```
for(int i=0;i<10;i++){
    int fd=open("testfifo",O_RDONLY);
    if(fd==-1){
        perror("Error opening");
    }
    int d;
    for(int j=0;j<5;j++){
        char *c=malloc(sizeof(char)*21);
        int e;
        if(read(fd,c,sizeof(char) *21)==-1){
            perror("Some error reading");
        }

        if(read(fd,&e,sizeof(int))==-1){
            perror("Some error reading");
        }
        printf("%s %d\n",c,e);
        if(j==4){
```

The overhead of opening and closing the same fifo same with blocking calls for fifo makes fifo take the most time to transfer 50 strings. However the code is extremely easy since there is no need to synchronize.

Output:





Notes:

1)Start ./Fifo2 only after ./Fifo else error will be raised.

2)Start ./socket2 only after ./socket1 else error will be raised.

3)Start ./s2 only after ./s else it will execute . ./s2 has to be done very quickly after ./s else there will be a Bus Error raised.