1. **IPC using pipe**

#include <sys/wait.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

int main(int argc, char \*argv[])

{

int pipefd[2];

pid\_t cpid;

char buf;

char inp[50];

printf("Enter input\n");

scanf("%[^\n]s",inp);

if (pipe(pipefd) == -1) {

perror("pipe");

exit(EXIT\_FAILURE);

}

cpid = fork();

if (cpid == -1) {

perror("fork");

exit(EXIT\_FAILURE);

}

if (cpid == 0) { /\* Child reads from pipe \*/

close(pipefd[1]);

printf("That is what the child process with id %d with parent id %d read \n",getpid(),getppid()) ; /\* Close unused write end \*/

int count=0;

while (read(pipefd[0], &buf, 1) > 0){

write(STDOUT\_FILENO, &buf, 1);

count++;

}

write(STDOUT\_FILENO, "\n", 1);

printf("These many bytes were written %d\n",count);

close(pipefd[0]);

\_exit(EXIT\_SUCCESS);

} else {

/\* Parent writes argv[1] to pipe \*/

printf("That is what the Parent process with id %d with parent id %d wrote \n%s\n",getpid(),getppid(),inp) ;

close(pipefd[0]); /\* Close unused read end \*/

write(pipefd[1], inp, strlen(inp));

close(pipefd[1]); /\* Reader will see EOF \*/

wait(NULL); /\* Wait for child \*/

exit(EXIT\_SUCCESS);

}

}

**Output:**

Enter input

Hello world !!

That is what the Parent process with id 4198 with parent id 3767 wrote

Hello world !!

That is what the child process with id 4199 with parent id 4198 read

Hello world !!

These many bytes were written 14

1. **IPC using FIFO**

**Server.c**

#include<stdio.h>

#include<unistd.h>

#include<fcntl.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<string.h>

int main()

{

char fname[25]="";

char fcontent[100]="";

int fd,fd1,fd2;

mkfifo("fifo1",0600);

mkfifo("fifo2",0600);

fd=open("fifo1",O\_RDONLY);

fd1=open("fifo2",O\_WRONLY);

read(fd,fname,25);

printf("Filename from Client: %s\n",fname);

fd2=open(fname,O\_RDONLY);

while(read(fd2,fcontent,100)!=0)

{

//printf("%s\n",fcontent);

if(fd<0)

write(fd1,"File does not exist !",14);

else

write(fd1,fcontent,strlen(fcontent));

}

close(fd);

close(fd1);

close(fd2);

}

**Client.c**

#include<stdio.h>

#include<unistd.h>

#include<fcntl.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<string.h>

int main()

{

char s[100]="";

char s1[1000]="";

int fd,fd1;

fd=open("fifo1",O\_WRONLY);

fd1=open("fifo2",O\_RDONLY);

printf("\nEnter Filename: ");

scanf("%s",s);

write(fd,s,strlen(s));

while(read(fd1,s1,1000)!=0)

{

printf("File Content:\n%s\n",s1);

}

}

**Output:**

**Test file:**

this is the test file.

reading..

**Server side:**

Filename from Client: test

**Client side :**

Enter Filename: test

File Content:

this is the test file.

reading..

**1.3 IPC using message queue**

**Server.c**

#include<stdio.h>

#include<unistd.h>

#include<fcntl.h>

#include<sys/types.h>

#include<sys/ipc.h>

#include<sys/msg.h>

#include<sys/stat.h>

#include<string.h>

#define KEY 500

struct msg

{

long int type;

char a[1024];

int pid;

}p;

int main()

{

int m,n,fd,m1;

m=msgget(KEY,0666|IPC\_CREAT);

while(1)

{

msgrcv(m,&p,sizeof(p),1,0);

printf("Filename from Client: %s\n",p.a);

fd=open(p.a,O\_RDONLY);

n=read(fd,p.a,1024);

p.type=p.pid;

p.pid=getpid();

msgsnd(m,&p,sizeof(p),0);

}

}

**Client.c**

#include<stdio.h>

#include<unistd.h>

#include<fcntl.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<string.h>

#define KEY 500

struct msg

{

long int type;

char a[1024];

int pid;

}p,p1;

int main()

{

int m;

m=msgget(KEY,0);

p.type=1;

printf("\nEnter Filename: ");

scanf("%s",p.a);

pid\_t pid;

p.pid=getpid();

msgsnd(m,&p,sizeof(p),0);

msgrcv(m,&p1,sizeof(p),p.pid,0);

printf("File Content:\n%s\n",p1.a);

}

**Output:**

Server:

Filename from Client: test

Client:

Enter Filename: test

File Content:

this is the test file.

reading..

**1.4 IPC using Shared memory**

**Server.c**

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

#define SHMSZ 27

main()

{

char c;

int shmid;

key\_t key;

char \*shm, \*s;

/\*

\* We'll name our shared memory segment

\* "5678".

\*/

key = 5678;

/\*

\* Create the segment.

\*/

if ((shmid = shmget(key, SHMSZ, IPC\_CREAT | 0666)) < 0) {

perror("shmget");

exit(1);

}

/\*

\* Now we attach the segment to our data space.

\*/

if ((shm = shmat(shmid, NULL, 0)) == (char \*) -1) {

perror("shmat");

exit(1);

}

/\*

\* Now put some things into the memory for the

\* other process to read.

\*/

s = shm;

for (c = 'a'; c <= 'z'; c++)

\*s++ = c;

\*s = NULL;

/\*

\* Finally, we wait until the other process

\* changes the first character of our memory

\* to '\*', indicating that it has read what

\* we put there.

\*/

while (\*shm != '\*')

sleep(1);

exit(0);

}

Client.c

/\*

\* shm-client - client program to demonstrate shared memory.

\*/

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

#define SHMSZ 27

main()

{

int shmid;

key\_t key;

char \*shm, \*s;

/\*

\* We need to get the segment named

\* "5678", created by the server.

\*/

key = 5678;

/\*

\* Locate the segment.

\*/

if ((shmid = shmget(key, SHMSZ, 0666)) < 0) {

perror("shmget");

exit(1);

}

/\*

\* Now we attach the segment to our data space.

\*/

if ((shm = shmat(shmid, NULL, 0)) == (char \*) -1) {

perror("shmat");

exit(1);

}

/\*

\* Now read what the server put in the memory.

\*/

for (s = shm; \*s != NULL; s++)

putchar(\*s);

putchar('\n');

/\*

\* Finally, change the first character of the

\* segment to '\*', indicating we have read

\* the segment.

\*/

\*shm = '\*';

exit(0);

}

**Output:**

Server wrote in shared Memory:

abcdefghijklmnopqrstuvwxyz

Client read from shared Memory:

abcdefghijklmnopqrstuvwxyz

**1.5 IPC using semaphores**

**Server.c**

#include<stdio.h>

#include<sys/types.h>

#include<sys/sem.h>

#include<unistd.h>

#include<sys/ipc.h>

#define KEY 1010

union semun {

int val;

struct semid\_ds \*buf;

unsigned short \*array;

struct seminfo \*\_\_buf;

};

void main()

{

int semid,count=0,fd;

struct sembuf put,get;

union semun u;

unsigned a[2]={1,0};

semid=semget(KEY,2,IPC\_CREAT|IPC\_EXCL|0666);

u.array=a;

semctl(semid,0,SETALL,u);

fd=open("buff",O\_CREAT|O\_RDWR,0666);

write(fd,&count,sizeof(int));

while(1)

{put.sem\_num=0;

put.sem\_op=-1;

put.sem\_flg=0;

semop(semid,&put,1);

lseek(fd,0,0);

read(fd,&count,sizeof(int));

count++;

lseek(fd,0,0);

write(fd,&count,sizeof(int));

printf("Produced item: %d\n",count);

get.sem\_num=1;

get.sem\_op=1;

get.sem\_flg=0;

semop(semid,&get,1);

sleep(5); }

close(fd);}

**Client.c**

#include<stdio.h>

#include<sys/types.h>

#include<sys/sem.h>

#include<unistd.h>

#include<sys/ipc.h>

#include<fcntl.h>

#define KEY 1010

union semun {

int val;

struct semid\_ds \*buf;

unsigned short \*array;

struct seminfo \*\_\_buf;

};

void main()

{ int semid,count,fd;

struct sembuf put,get;

semid=semget(KEY,0,0);

fd=open("buff",O\_RDONLY);

while(1)

{get.sem\_num=1;

get.sem\_op=-1;

get.sem\_flg=0;

semop(semid,&put,1);

lseek(fd,0,0);

read(fd,&count,sizeof(int));

printf("Consumed item: %d\n",count);

put.sem\_num=0;

put.sem\_op=1;

put.sem\_flg=0;

semop(semid,&put,1);

sleep(5);}

close(fd); }

**Outputs:**

**Server:**

Produced item: 1

Produced item: 2

Produced item: 3

**Client:**

Consumed item: 1

Consumed item: 2

Consumed item: 3

1. **CLIENT-SERVER using sockets**

**Server.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h> //for perror

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <sys/wait.h>

#include <signal.h>

#define PORT "3490" // the port users will be connecting to

#define BACKLOG 10 // how many pending connections queue will hold

// get sockaddr, IPv4 or IPv6:

void \*get\_in\_addr(struct sockaddr \*sa)

{

if (sa->sa\_family == AF\_INET) {

return &(((struct sockaddr\_in\*)sa)->sin\_addr);

}

return &(((struct sockaddr\_in6\*)sa)->sin6\_addr);

}

int main(void)

{

int sockfd, new\_fd; // listen on sock\_fd, new connection on new\_fd

struct addrinfo hints, \*servinfo, \*p; //servinfo will point to hints

struct sockaddr\_storage their\_addr; // connector's address information

socklen\_t sin\_size;

int yes=1;

char buf[100];

char s[INET6\_ADDRSTRLEN]; //#define in netinet/in.h Value is 46.INET\_ADDRSTRLEN 16

int rv;

memset(&hints, 0, sizeof hints);

hints.ai\_family = AF\_UNSPEC; //IP can be either IPv4 or IPv6

hints.ai\_socktype = SOCK\_STREAM; // Stream socket

hints.ai\_flags = AI\_PASSIVE; // use my IP.attach IP of host to structure

if ((rv = getaddrinfo(NULL, PORT, &hints, &servinfo)) != 0) //servinfo stores info of all IP associate with host

{

fprintf(stderr, "getaddrinfo: %s\n", gai\_strerror(rv));

return 1;

}

// loop through all the results and bind to the first we can

for(p = servinfo; p != NULL; p = p->ai\_next)

{

if ((sockfd = socket(p->ai\_family, p->ai\_socktype,p->ai\_protocol)) == -1) //Socket Connection is being established

{

perror("server: socket");

continue;

}

if (bind(sockfd, p->ai\_addr, p->ai\_addrlen) == -1) //binding to the host

{

close(sockfd);

perror("server: bind");

continue;

}

break;

}

if (p == NULL)

{

fprintf(stderr, "server: failed to bind\n");

return 2;

}

freeaddrinfo(servinfo); // all done with this structure

if (listen(sockfd, BACKLOG) == -1)

{

perror("listen");

exit(1);

}

printf("server: waiting for connections...\n");

int num;

while(1)

{ // main accept() loop

sin\_size = sizeof their\_addr;

new\_fd = accept(sockfd, (struct sockaddr \*)&their\_addr, &sin\_size);

if (new\_fd == -1)

{

perror("accept");

continue;

}

inet\_ntop(their\_addr.ss\_family,get\_in\_addr((struct sockaddr \*)&their\_addr),s, sizeof s);

char Msg[100];

printf("server: got connection from %s\n", s);

while(1)

{

num = recv(new\_fd,buf,sizeof(buf),0);

buf[num] = '\0';

if(strcmp(buf,"close") == 0)

{

printf("Connection closed\n");

return 0;

}

else

printf("Message received from client : %s\n",buf);

printf("Enter Message to be send: ");

gets(Msg);

send(new\_fd, Msg, sizeof(Msg), 0);

}

close(new\_fd); // parent doesn't need this

}

return 0;

}

**Client.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <netdb.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#define PORT "3490" // the port client will be connecting to

#define MAXDATASIZE 100 // max number of bytes we can get at once

// get sockaddr, IPv4 or IPv6:

void \*get\_in\_addr(struct sockaddr \*sa)

{

if (sa->sa\_family == AF\_INET) {

return &(((struct sockaddr\_in\*)sa)->sin\_addr);

}

return &(((struct sockaddr\_in6\*)sa)->sin6\_addr);

}

int main(int argc, char \*argv[])

{

int sockfd, numbytes;

char buf[MAXDATASIZE];

struct addrinfo hints, \*servinfo, \*p;

int rv;

char s[INET6\_ADDRSTRLEN];

if (argc != 2) {

fprintf(stderr,"usage: client hostname\n");

exit(1);

}

memset(&hints, 0, sizeof hints);

hints.ai\_family = AF\_UNSPEC;

hints.ai\_socktype = SOCK\_STREAM;

if ((rv = getaddrinfo(argv[1], PORT, &hints, &servinfo)) != 0)

{

fprintf(stderr, "getaddrinfo: %s\n", gai\_strerror(rv));

return 1;

}

// loop through all the results and connect to the first we can

for(p = servinfo; p != NULL; p = p->ai\_next)

{

if ((sockfd = socket(p->ai\_family, p->ai\_socktype,p->ai\_protocol)) == -1)

{

perror("client: socket");

continue;

}

if (connect(sockfd, p->ai\_addr, p->ai\_addrlen) == -1)

{

close(sockfd);

perror("client: connect");

continue;

}

break;

}

if (p == NULL)

{

fprintf(stderr, "client: failed to connect\n");

return 2;

}

inet\_ntop(p->ai\_family, get\_in\_addr((struct sockaddr \*)p->ai\_addr),s, sizeof s);

printf("client: connecting to %s\n", s);

freeaddrinfo(servinfo); // all done with this structure

char Msg[100];

while(1)

{

//Client Sending to Server.

printf("Enter message to be sent to server : ");

gets(Msg);

send(sockfd, Msg, strlen(Msg), 0);

if(strcmp(Msg,"close") == 0)

return 0;

//Server sending to Client.

start :

numbytes = recv(sockfd, buf, MAXDATASIZE-1, 0);

buf[numbytes] = '\0';

if(!buf[0])

goto start;

printf("client received : %s\n",buf);

}

//printf("close\n");

close(sockfd);

return 0;

}

**Output:**

**Server:**

server: waiting for connections...

server: got connection from 120.11.3.0

Message received from client : Hello server

Enter Message to be send: hi client!

Message received from client : Bye

**Client:**

client: connecting to 120.11.3.0

Enter message to be sent to server : Hello server

client received : hi client!

Enter message to be sent to server : Bye

close

Connection closed