EXP1 : Pipes:-

#include <iostream>

#include <stdio.h>

#include <unistd.h>

#include <string.h>

#define MAX 100

using namespace std;

int main()

{

int status,pid;

int pipes[2];

char name[MAX];

cout<<"Enter message to send: ";

gets(name);

char str[100];

status = pipe(pipes);

if(status== -1)

{

cout<<"Error";

return 0;

}

pid = fork();

if(pid == -1)

{

cout<<"Error";

return 0;

}

//1 - write descriptor

//0 - read descriptor

else if(pid== 0)

{

close(pipes[0]);

cout<<"\nSending message...\n";

write(pipes[1], name, MAX );

close(pipes[1]);

}

else

{

close(pipes[1]);

read(pipes[0], str, MAX);

cout<<"\nRecieved\n";

cout<<str<<endl;

close(pipes[0]);

}

return 0;

}

EXP2: Named Pipe FIFO :-

Reader.c

#include <fcntl.h>

#include <stdio.h>

#include <sys/stat.h>

#include <unistd.h>

#include <string.h>

#define MAX 100

int main()

{

int fd;

char \* myfifo = "/tmp/myfifo";

char str[MAX];

/\* open, read, and display the message from the FIFO \*/

fd = open(myfifo, O\_RDONLY);

read(fd, str, MAX);

printf("Received: %s\n", str);

close(fd);

return 0;

}

WRITER:-

#include <fcntl.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <unistd.h>

#include <stdio.h>

int main()

{

int fd; int status;

char \* myfifo = "/tmp/myfifo";

//defines the file name of the fifo we want to create

status = mkfifo(myfifo, 0666);

//0666 is the mode of the fifo, which defines the permissions

/\*status = 0 means failure

if(status==-1)

{

printf("Error\n");

return 0;

}\*/

char str[100];

printf("Enter message to send: ");

gets(str);

fd = open(myfifo, O\_WRONLY);

write(fd, str, sizeof(str));

close(fd);

/\* remove the FIFO \*/

unlink(myfifo);

return 0;

}

EXP 3: Message Queue:-

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;

int t;

cout<<"Enter number of messages: ";

cin>>t;

while(t--)

{

printf("\nEnter message: ");

gets(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);

return 0;

}

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("Message 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);

}

return 0;

}

EXP:4 Shared Memory

SharedMem\_sender

/\*

Shared is the fastest medium of IPC communication. It's doesn't require any intermediate structures to communicate.

Shared memory segments exist in the kernel's address space only.

shmget() system call is used to create a new shared memory.

Returns the shared memory segment identifier on success, and -1 on error.

First argument - key value

Second argument - requested segment size

Third argument shmflag which can be used to set the creating control flags like IPC\_CREAT

shmat() is used to attach a shared memory to a process address space.

It returns a pointer 'shmaddr' to the head of the shared segment.

First argument -> shmid

Second argument -> shmaddr, usually kept 0. ) means that the system will select the starting address for the caller.

Third argument -> shmflg

shmdt() detaches the shared memory segment located at the address indicated by shmaddr.

This doesn't remove the segment from the kernel!

\*/

/\*

\*\* shmdemo.c -- read and write to a shared memory segment

\*/

/\*

\*\* shmdemo.c -- read and write to a shared memory segment

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#define SHM\_SIZE 1024 /\* make it a 1K shared memory segment \*/

int main()

{

key\_t key;

int shmid;

char \*data;

int mode;

/\* make the key: \*/

if ((key = ftok("sharedMem.c", 'R')) == -1) {

perror("ftok");

exit(1);

}

/\* connect to (and possibly create) the segment: \*/

if ((shmid = shmget(key, SHM\_SIZE, 0644 | IPC\_CREAT)) == -1) {

perror("shmget");

exit(1);

}

char buff[100];

/\* attach to the segment to get a pointer to it: \*/

data = shmat(shmid, (void \*)0, 0);

if (data == (char \*)(-1)) {

perror("shmat");

exit(1);

}

char ch = 'y';

while(ch == 'y' || ch=='Y')

{

printf("Enter message: ");

scanf("%s",buff);

printf("writing to segment...\n");

strncpy(data, buff, SHM\_SIZE);

printf("Wish to enter more?(y/n): ");

fflush(stdin);

scanf("%s",&ch);

}

/\* detach from the segment: \*/

if (shmdt(data) == -1) {

perror("shmdt");

exit(1);

}

return 0;

}

SharedMem\_reciever:-

/\*

\*\* shmdemo.c -- read and write to a shared memory segment

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#define SHM\_SIZE 1024 /\* make it a 1K shared memory segment \*/

int main()

{

key\_t key;

int shmid;

char \*data;

int mode;

/\* make the key: \*/

if ((key = ftok("sharedMem.c", 'R')) == -1) {

perror("ftok");

exit(1);

}

/\* connect to (and possibly create) the segment: \*/

if ((shmid = shmget(key, SHM\_SIZE, 0644 | IPC\_CREAT)) == -1) {

perror("shmget");

exit(1);

}

/\* attach to the segment to get a pointer to it: \*/

data = shmat(shmid, (void \*)0, 0);

if (data == (char \*)(-1)) {

perror("shmat");

exit(1);

}

char ch='y';

while(ch=='y')

{

printf("segment contains: \"%s\"\n", data);

printf("Wish to Coninue?(y/n)\n");

scanf("%s",&ch);

}

/\* detach from the segment: \*/

if (shmdt(data) == -1) {

perror("shmdt");

exit(1);

}

return 0;

}

EXP:5 Semaphores

Client :-

#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);

}

Server:-

#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=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);

}

RPC program:-

msg.x  
  
program MSG\_PROG {

version MSG\_VERS {

string MSG(string) = 1;

} = 1;

} = 0x23451444;  
  
  
  
msg\_server

#include "msg.h"

char \*\*

msg\_1\_svc(char \*\*argp, struct svc\_req \*rqstp)

{

static char \* result;

char \*temp;

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

printf("Enter reply : ");

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

result=temp;

return &result;

}

msg\_client :-

#include "msg.h"

void

msg\_prog\_1(char \*host)

{

CLIENT \*clnt;

char \* \*result\_1;

char \* msg\_1\_arg;

char\* temp;

temp=malloc(500);

#ifndef DEBUG

clnt = clnt\_create (host, MSG\_PROG, MSG\_VERS, "udp");

if (clnt == NULL) {

clnt\_pcreateerror (host);

exit (1);

}

else printf("Established connection with %s server using udp protocol\n\n",host);

#endif /\* DEBUG \*/

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

scanf("%[^\n]499s",temp);

msg\_1\_arg = temp;

result\_1 = msg\_1(&msg\_1\_arg, clnt);

if (result\_1 == (char \*\*) NULL) {

clnt\_perror (clnt, "call failed");

}

else

printf("\nMessage successfully delivered!\nReply received : %s\n", \*result\_1);

#ifndef DEBUG

clnt\_destroy (clnt);

#endif /\* DEBUG \*/

}

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

{

char \*host;

if (argc < 2) {

printf ("usage: %s server\_host\n", argv[0]);

exit (1);

}

host = argv[1];

msg\_prog\_1 (host);

exit (0);

}