## task1:

step1:

#include <stdio.h>

#include <unistd.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <error.h>

#define SIZE 1024

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

{

int shmid ;

char \*shmaddr ;

struct shmid\_ds buf ;

int flag = 0 ;

int pid ;

shmid = shmget(IPC\_PRIVATE, SIZE, IPC\_CREAT|0600 ) ;

if ( shmid < 0 )

{

perror("get shm ipc\_id error") ;

return -1 ;

}

printf("shmid is: %d\n", shmid);

return 0 ;

}

step2:

#include <stdio.h>

#include <unistd.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <error.h>

#define SIZE 1024

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

{

int shmid ;

char \*shmaddr ;

struct shmid\_ds buf ;

int flag = 0 ;

int pid ;

shmid = shmget(IPC\_PRIVATE, SIZE, IPC\_CREAT|0600 ) ;

if ( shmid < 0 )

{

perror("get shm ipc\_id error") ;

return -1 ;

}

pid = fork() ;

if ( pid == 0 )

{

printf("this is child.\n");

return 0;

} else if ( pid > 0) {

wait(NULL);

printf("this is parent.\n");

}else{

perror("fork error") ;

shmctl(shmid, IPC\_RMID, NULL) ;

}

return 0 ;

}

step3:

#include <stdio.h>

#include <unistd.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <error.h>

#define SIZE 1024

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

{

int shmid ;

char \*shmaddr ;

struct shmid\_ds buf ;

int flag = 0 ;

int pid ;

char tmpbuf[SIZE];

shmid = shmget(IPC\_PRIVATE, SIZE, IPC\_CREAT|0600 ) ;

if ( shmid < 0 )

{

perror("get shm ipc\_id error") ;

return -1 ;

}

pid = fork() ;

if ( pid == 0 )

{

shmaddr = (char \*)shmat( shmid, NULL, 0 ) ;

if ( (intptr\_t)shmaddr == -1 )

{

perror("shmat addr error") ;

return -1 ;

}

printf("please input something:\n");

fgets(tmpbuf,SIZE,stdin);

strncpy( shmaddr, tmpbuf,strlen(tmpbuf)) ;

printf("this is child.\ninput info to shm.\nyour input string is : %s\n", shmaddr);

shmdt((const void \*)shmaddr ) ;

return 0;

} else if ( pid > 0) {

wait(NULL);

// printf("\nthis is parent.\n");

}else{

perror("fork error") ;

shmctl(shmid, IPC\_RMID, NULL) ;

}

return 0 ;

}

step4:

#include <stdio.h>

#include <unistd.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <error.h>

#define SIZE 1024

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

{

int shmid ;

char \*shmaddr ;

struct shmid\_ds buf ;

int flag = 0 ;

int pid ;

shmid = shmget(IPC\_PRIVATE, SIZE, IPC\_CREAT|0600 ) ;

if ( shmid < 0 )

{

perror("get shm ipc\_id error") ;

return -1 ;

}

pid = fork() ;

if ( pid == 0 )

{

shmaddr = (char \*)shmat( shmid, NULL, 0 ) ;

if ( (intptr\_t)shmaddr == -1 )

{

perror("shmat addr error") ;

return -1 ;

}

strcpy( shmaddr, argv[1]) ;

printf("this is child.\nwrite argv[1] to shm.\nyour input string is \'%s\'\n", shmaddr);

shmdt( shmaddr ) ;

return 0;

} else if ( pid > 0) {

wait(NULL);

printf("this is parent.\n");

shmctl(shmid,IPC\_STAT,&buf);

printf("the size of new buf is %d bytes %d\n", (int)buf.shm\_segsz,(int)buf.shm\_cpid) ;

printf("parent pid is %d\n", getpid()) ;

printf("chlid pid is %d\n",pid) ;

}else{

perror("fork error") ;

shmctl(shmid, IPC\_RMID, NULL) ;

}

return 0 ;

}

step5:

#include <stdio.h>

#include <unistd.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <error.h>

#define SIZE 1024

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

{

int shmid ;

char \*shmaddr ;

struct shmid\_ds buf ;

int flag = 0 ;

int pid ;

char tmpbuf[SIZE];

shmid = shmget(IPC\_PRIVATE, SIZE, IPC\_CREAT|0600 ) ;

if ( shmid < 0 )

{

perror("get shm ipc\_id error") ;

return -1 ;

}

pid = fork() ;

if ( pid == 0 )

{

shmaddr = (char \*)shmat( shmid, NULL, 0 ) ;

if ( (intptr\_t)shmaddr == -1 )

{

perror("shmat addr error") ;

return -1 ;

}

printf("please input something:\n");

fgets(tmpbuf,SIZE,stdin);

strncpy( shmaddr, tmpbuf,strlen(tmpbuf)) ;

//printf("this is child.\ninput sth to shm.\nyour input string is : %s\n", shmaddr);\*/

shmdt( shmaddr ) ;

return 0;

} else if ( pid > 0) {

wait(NULL);

flag = shmctl( shmid, IPC\_STAT, &buf) ;

if ( flag == -1 )

{

perror("shmctl shm error") ;

return -1 ;

}

//printf("\nthis is parent.\n");

printf("the size of shared memory is %d bytes\n", buf.shm\_segsz ) ;

// printf("parent pid is %d\n", getpid()) ;

//printf("chlid pid is %d\n",pid) ;

shmaddr = (char \*) shmat(shmid, NULL, 0 ) ;

if ( (intptr\_t)shmaddr == -1 )

{

perror("shmat addr error") ;

return -1 ;

}

printf("your input string is : %s", shmaddr) ;

shmdt( shmaddr ) ;

shmctl(shmid, IPC\_RMID, NULL) ;

}else{

perror("fork error") ;

shmctl(shmid, IPC\_RMID, NULL) ;

}

return 0 ;

}

## task2:

read.c

#include <stdio.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/types.h>

#include <unistd.h>

typedef struct{

char no[8];

char name[8];

double price;

}book;

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

{

int shm\_id,i;

key\_t key;

book \*shmaddr;

char pathname[30] ;

strcpy(pathname,"/tmp") ;

key = ftok(pathname,0x03);

if(key == -1)

{

perror("ftok error");

return -1;

}

printf("key=%d\n", key) ;

shm\_id = shmget(key,0, 0);

if(shm\_id == -1)

{

perror("shmget error");

return -1;

}

printf("shm\_id=%d\n", shm\_id) ;

shmaddr = (book\*)shmat(shm\_id,NULL,0);

printf("Book Information:\n");

for(i = 0;i<2;i++)

{

printf( "no:%s",(\*(shmaddr+i)).no );

printf( " name:%s",(\*(shmaddr+i)).name );

printf( " price:%f\n",(\*(shmaddr+i)).price );

}

if(shmdt(shmaddr) == -1)

{

perror("detach error");

return -1;

}

if(shmctl(shm\_id,IPC\_RMID,NULL) == -1){

perror("shmctl error");

return -1;

}

return 0 ;

}

write.c:

#include <stdio.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/types.h>

#include <unistd.h>

#include <string.h>

#define SIZE 1024

typedef struct{

char no[8];

char name[8];

double price;

}book;

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

{

int shm\_id,i;

key\_t key;

char temp[8];

book \*shmaddr;

char pathname[30] ;

book bookInfo[2];

strcpy(bookInfo[0].no,"01");

strcpy(bookInfo[0].name,"book1");

strcpy(bookInfo[1].no,"02");

strcpy(bookInfo[1].name,"book2");

bookInfo[0].price = 10;

bookInfo[1].price = 20;

strcpy(pathname,"/tmp") ;

key = ftok(pathname,0x03);

if(key==-1)

{

perror("ftok error");

return -1;

}

printf("key=%d\n",key) ;

shm\_id=shmget(key,SIZE,IPC\_CREAT|IPC\_EXCL|0600);

if(shm\_id==-1)

{

perror("shmget error");

return -1;

}

printf("shm\_id=%d\n", shm\_id) ;

shmaddr=(book\*)shmat(shm\_id,NULL,0);

for(i = 0;i<2;i++)

{

strncpy((shmaddr+i)->no,bookInfo[i].no,sizeof(bookInfo[i].no));

strncpy((shmaddr+i)->name,bookInfo[i].name,sizeof(bookInfo[i].name));

(shmaddr+i)->price = bookInfo[i].price;

}

shmdt(shmaddr) ;

return 0 ;{}

}

myanswer:

read.c:

#include <stdio.h>

#include <stdlib.h>

#include <sys/shm.h>

#include <sys/stat.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/types.h>

#include <unistd.h>

#define PERM IPC\_CREAT|S\_IRUSR|S\_IWUSR

#define SIZE 1024

typedef struct{

char no[8];

char name[8];

double price;

}Book\_struc;

int main()

{

Book\_struc \*preadBook;

Book\_struc \*paddr=NULL,\*paddr2=NULL;

key\_t key;

int shm\_id;

key=ftok("tmp",0x03);

if(key<0)

{

perror("ftok error");

return -1;

}

printf("key=%d\n",key);

shm\_id=shmget(key,0,0);

if(shm\_id<0)

{

perror("creat shm error");

return -1;

}

printf("shm\_id is %d\n",shm\_id);

paddr=(Book\_struc \*)shmat(shm\_id,NULL,0);

if((intptr\_t)paddr ==-1)

{

perror("shmat error");

return -1;

}

printf("book1 is \n");

printf("%s %s %f \n",paddr->no,paddr->name,paddr->price);

paddr2=paddr+sizeof(Book\_struc);

printf("book2 is \n");

printf("%s %s %f \n",paddr2->no,paddr2->name,paddr2->price);

if(shmdt(paddr)==-1)

{

perror("detach error");

return -1;

}

if( shmctl(shm\_id,IPC\_RMID,NULL)<0)

{

perror("shmctl error");

return -1;

}

return 0;

}

write.c:

#include <stdio.h>

#include <stdlib.h>

#include <sys/shm.h>

#include <sys/stat.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/types.h>

#include <unistd.h>

#define PERM IPC\_CREAT|S\_IRUSR|S\_IWUSR

#define SIZE 1024

typedef struct{

char no[8];

char name[8];

double price;

}Book\_struc;

int main()

{

Book\_struc book1={0},book2={0};

Book\_struc \*paddr=NULL,\*paddr2=NULL;

key\_t key;

int shm\_id;

strcpy((char \*)book1.no,"01");

strcpy((char \*)book1.name,"book1");

book1.price=54.5;

strcpy(book2.no,"02");

strcpy(book2.name,"book2");

book2.price=64.5;

key=ftok("tmp",0x03);

if(key<0)

{

perror("ftok error");

return -1;

}

printf("key=%d\n",key);

shm\_id=shmget(key,SIZE,PERM);

if(shm\_id<0)

{

perror("creat shm error");

return -1;

}

printf("shm\_id is %d\n",shm\_id);

paddr=(Book\_struc \*)shmat(shm\_id,NULL,0);

if((intptr\_t)paddr ==-1)

{

perror("shmat error");

return -1;

}

memset(paddr,0,SIZE);

memcpy(paddr,(const char \*)&book1,sizeof(Book\_struc));

printf("now book1 is\n");

printf("%s %s %f\n",paddr->no,paddr->name,paddr->price);

paddr2=paddr+sizeof(Book\_struc);

memcpy(paddr+sizeof(book1),(const char \*)&book2,sizeof(book2));

printf("now book2 is\n");

printf("%s %s %f\n",paddr2->no,paddr2->name,paddr2->price);

shmdt(paddr);

}

## task3:

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#define SIZE 128

int main()

{

int fd[2];

pid\_t pid;

char szout[SIZE];

char szin[SIZE];

printf("input str:\n");

fgets(szin,100,stdin);

pipe(fd);

while((pid=fork())<0);

if(pid==0)

{

close(fd[1]);

read(fd[0],szout,SIZE);

fprintf(stdout,"I am child,out:%s\n",szout);

exit(0);

}

else

{

close(fd[0]);

write(fd[1],szin,SIZE);

printf("I am parent,in:%s\n",szin);

sleep(1);

}

return 0;

}

## task4:

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>

#define SIZE 128

int main()

{

int fd[2];

pid\_t pid;

char szout[SIZE];

char szin[SIZE];

memset((void \*)szout,0,(size\_t)SIZE);

memset((void \*)szin,0,(size\_t)SIZE);

printf("input arg:\n");

scanf("%s",szin);

pipe(fd);

pid=fork();

if(pid==0)

{

close(fd[1]);

read(fd[0],szout,SIZE);

execl("/bin/ls","ls",(const char \*)szout,NULL);

sleep(2);

exit(0);

}

else if(pid>0)

{

close(fd[0]);

write(fd[1],szin,SIZE);

sleep(3);

exit(0);

}

return 0;

}

## task5:

read.c

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <fcntl.h>

#include <limits.h>

#include <sys/types.h>

#include <sys/stat.h>

#define FIFO\_NAME "/tmp/my\_fifo"

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

{

int pipe\_fd;

int res;

char buffer[4096];

int bytes\_read = 0;

memset(buffer, '\0', sizeof(buffer));

printf("Process %d opening FIFO O\_RDONLY\n", getpid());

pipe\_fd = open(FIFO\_NAME, O\_RDONLY);

printf("process read:the file's descriptor is %d\n", pipe\_fd);

if(pipe\_fd != -1)

{

bytes\_read = read(pipe\_fd, buffer, sizeof(buffer));

printf("the read data is %s\n", buffer);

close(pipe\_fd);

}

else

{

exit(EXIT\_FAILURE);

}

printf("Process %d finished, %d bytes read\n", getpid(), bytes\_read);

exit(EXIT\_SUCCESS);

}

write.c

#include<unistd.h>

#include<stdlib.h>

#include<stdio.h>

#include<string.h>

#include<fcntl.h>

#include<limits.h>

#include<sys/types.h>

#include<sys/stat.h>

#define FIFO\_NAME "/tmp/my\_fifo"

int main(int argc,char \*agrv[]){

int pipe\_fd;

int res;

char buffer[] = "hello world!";

if(access(FIFO\_NAME,F\_OK) == -1)

{

res = mkfifo(FIFO\_NAME,0766);

if(res != 0)

{

fprintf(stderr,"Could not create fifo %s\n",FIFO\_NAME);

exit(EXIT\_FAILURE);

}

}

printf("Process %d opening FIFO O\_WRONLY\n",getpid());

pipe\_fd = open(FIFO\_NAME,O\_WRONLY);

printf("process write:the file's descriptor is %d\n",pipe\_fd);

if(pipe\_fd != -1)

{

res = write(pipe\_fd,buffer,sizeof(buffer));

if(res == -1)

{

fprintf(stderr,"Write error on pipe\n");

exit(EXIT\_FAILURE);

}

printf("Write date is %s, %d bytes is write\n",buffer,res);

(void)close(pipe\_fd);

}

else

{

exit(EXIT\_FAILURE);

}

printf("Process %d finished\n",getpid());

exit(EXIT\_FAILURE);

return 0;

}

## task6:

#include <stdio.h>

#include <unistd.h>

#include <time.h>

#include <string.h>

#include <stdlib.h>

#include <sys/sem.h>

#include <sys/shm.h>

#include <sys/stat.h>

#define REPEATS (5) /\* count of production/consumption \*/

#define MAX\_BUFFER\_SIZE (3)

typedef struct

{

int bottom;

int top;

int data[MAX\_BUFFER\_SIZE];

} STRUCT\_BUFFER;

STRUCT\_BUFFER \* pBuffer = NULL;

/\* Define speed of consumer/producer, change them as u like \*/

#define PRODUCER\_SPEED (1) /\* 1/sec \*/

#define CONSUMER\_SPEED (5) /\* 1/2sec \*/

int sem\_consume; /\* consumer sem \*/

int sem\_produce; /\* producer sem \*/

int shm\_buffer; /\* shared buffer \*/

#define FLAG (IPC\_CREAT | S\_IRWXU)

/\* Init semphores & shared buffer \*/

void init()

{

union semun

{

int val;

struct semid\_ds \*buf;

unsigned short \*array;

} arg;

shm\_buffer = shmget(0x1111, sizeof(STRUCT\_BUFFER), FLAG);

pBuffer = shmat(shm\_buffer, 0, 0);

memset(pBuffer, 0, sizeof(STRUCT\_BUFFER));

sem\_consume = semget(0x2222, 1, FLAG);

arg.val = 0;

if (semctl(sem\_consume, 0, SETVAL, arg) < 0)

{

perror("Consumer");

exit(1);

}

sem\_produce = semget(0x3333, 1, FLAG);

arg.val = MAX\_BUFFER\_SIZE;

if (semctl(sem\_produce, 0, SETVAL, arg) < 0)

{

perror("Producer");

exit(1);

}

}

/\* destroy semphores & shared buffer \*/

void deinit()

{

shmctl(shm\_buffer, IPC\_RMID, NULL);

semctl(sem\_consume, 0, IPC\_RMID);

semctl(sem\_produce, 0, IPC\_RMID);

}

int main()

{

int pid, i;

struct sembuf sbuf;

init();

pid = fork();

if (pid > 0)

{

/\* parent process, consumer \*/

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

{

/\* Try decrementing 1 from consumer \*/

sbuf.sem\_num=0;

sbuf.sem\_op=-1;

sbuf.sem\_flg=0;

semop(sem\_consume, &sbuf, 1);//改变信号量的值

/\* OK \*/

system("date |awk '{print $5}' ");

printf("Consumer get %6d,pos=%d\n", pBuffer->data[pBuffer->bottom],pBuffer->bottom);

pBuffer->bottom = (pBuffer->bottom+1)%MAX\_BUFFER\_SIZE;

/\* Try incrementing 1 to producer \*/

sbuf.sem\_op = 1;

semop(sem\_produce, &sbuf, 1);

sleep(CONSUMER\_SPEED);

}

wait(0);

shmdt(pBuffer);

}

else if (pid == 0)

{

//srand(time(NULL));

//printf("time is %ld\n",time(NULL));

/\* child process, producer \*/

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

{

/\* Try decrementing 1 from producer \*/

sbuf.sem\_num=0;

sbuf.sem\_op=-1;

sbuf.sem\_flg=0;

semop(sem\_produce, &sbuf, 1);

/\* OK \*/

pBuffer->data[pBuffer->top] = (rand()%100) + i + 1;

system("date |awk '{print $5}' ");

printf("Producer put %6d,pos=%d\n", pBuffer->data[pBuffer->top],pBuffer->top);

pBuffer->top = (pBuffer->top+1)%MAX\_BUFFER\_SIZE;

/\* Try incrementing 1 to consumer \*/

sbuf.sem\_op = 1;

semop(sem\_consume, &sbuf, 1);

sleep(PRODUCER\_SPEED);

}

shmdt(pBuffer);

exit(0);

}

deinit();

return 0;

}

## Task7:

Receive.c

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <errno.h>

#include <sys/msg.h>

struct msg\_st

{

long int msg\_type;

char text[BUFSIZ];

};

int main()

{

int running = 1;

int msgid = -1;

struct msg\_st data;

long int msgtype = 0; //注意1

//建立消息队列

msgid = msgget((key\_t)1234, 0666 | IPC\_CREAT);

if(msgid == -1)

{

fprintf(stderr, "msgget failed with error: %d\n", errno);

exit(EXIT\_FAILURE);

}

//从队列中获取消息，直到遇到end消息为止

while(running)

{

if(msgrcv(msgid, (void\*)&data, BUFSIZ, msgtype, 0) == -1)

{

fprintf(stderr, "msgrcv failed with errno: %d\n", errno);

exit(EXIT\_FAILURE);

}

printf("You wrote: %s\n",data.text);

//遇到end结束

if(strncmp(data.text, "end", 3) == 0)

running = 0;

}

//删除消息队列

if(msgctl(msgid, IPC\_RMID, 0) == -1)

{

fprintf(stderr, "msgctl(IPC\_RMID) failed\n");

exit(EXIT\_FAILURE);

}

exit(EXIT\_SUCCESS);

}

Send.c

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <sys/msg.h>

#include <errno.h>

#define MAX\_TEXT 512

struct msg\_st

{

long int msg\_type;

char text[MAX\_TEXT];

};

int main()

{

int running = 1;

struct msg\_st data;

char buffer[BUFSIZ];

int msgid = -1;

//建立消息队列

msgid = msgget((key\_t)1234, 0666 | IPC\_CREAT);

if(msgid == -1)

{

fprintf(stderr, "msgget failed with error: %d\n", errno);

exit(EXIT\_FAILURE);

}

//向消息队列中写消息，直到写入end

while(running)

{

//输入数据

printf("Enter some text: ");

fgets(buffer, BUFSIZ, stdin);

data.msg\_type = 1; //注意2

strcpy(data.text, buffer);

//向队列发送数据

if(msgsnd(msgid, (void\*)&data, MAX\_TEXT, 0) == -1)

{

fprintf(stderr, "msgsnd failed\n");

exit(EXIT\_FAILURE);

}

//输入end结束输入

if(strncmp(buffer, "end", 3) == 0)

running = 0;

sleep(1);

}

exit(EXIT\_SUCCESS);

}

## //附加题：

/\*  
\*由消息队列实现的简单聊天室程序(server)  
\*在服务器端主要实现广播功能，以及服务器退出以后通知  
\*所有客户端退出并删除消息队列功能  
\*对所有客户端的统计由链表实现  
\* \*/  
#include <stdio.h>  
#include <sys/types.h>  
#include <sys/ipc.h>  
#include <sys/msg.h>  
#include <stdlib.h>  
#include <errno.h>  
#include <string.h>  
#include <unistd.h>  
#include <sys/wait.h>  
#include <signal.h>  
#define TYPE\_SRV 10   
#define N 128  
struct msgbuf{  
    long mtype;  
    int subtype;  
    int pid;  
    char nick\_name[N];  
    char mtext[N];  
};  
#define MSG\_LEN (sizeof(struct msgbuf) - sizeof(long))  
typedef struct \_node\_{  
    int data;  
    struct \_node\_ \*next;  
}listnode,\*linklist;  
//function declare  
void Broadcast(linklist h,int msgid,struct msgbuf \*msg);  
linklist CreateLinklist(); //create linklist  
int Length(linklist h);     //get the linklist's length  
int Empty(linklist h);      //judge if the linklist is empty  
int Insert(linklist h,int value);  
void Delete(linklist h,int value);  
int GetValue(linklist h,int pos);  
int main(int argc,char \*argv[])  
{  
    key\_t key;  
    int msgid;  
    pid\_t pid;  
    struct msgbuf msg;  
    //生成key  
    if((key = ftok("./",'s')) < 0){  
        perror("ftok error.");  
        exit(-1);  
    }  
    //服务器进程创建创建消息队列  
    if((msgid = msgget(key,IPC\_CREAT|0666)) < 0){  
        perror("msgget error.");  
        exit(-1);  
    }  
    //使用fork创建子进程  
    if((pid = fork()) < 0){  
        perror("error to fork.");  
        exit(-1);  
    }  
    else if(pid == 0){//子进程专门用来实现消息的转发功能  
        //创建单链表  
        linklist h;  
        h = CreateLinklist();  
        while(1)  
        {  
            if(msgrcv(msgid,&msg,MSG\_LEN,TYPE\_SRV,0) < 0)  
            {  
                perror("msgrcv error.\n");  
                exit(1);  
            }  
            //对接收到的子进程进行子类型判断  
            switch(msg.subtype)  
            {  
            case 1 :    //register  
                if(Insert(h,msg.pid) != 0){  
                    printf("insert error.\n");  
                    exit(-1);  
                }  
                printf("[Name -%s- ID -%d- regist]\n",msg.nick\_name,msg.pid);  
                Broadcast(h,msgid,&msg);  
                break;  
            case 2 :    //broadcast   
                Broadcast(h,msgid,&msg);  
                //若收到console来的quit要求，在广播完以后，自己退出(不然会出现无限循环)  
                if(strncmp(msg.mtext,"quit",4)==0){  
                    exit(0);  
                }  
                break;  
            case 3 :    //unregister  
                if(!Empty(h)){  
                    Delete(h,msg.mtype);  
                }  
                  
                printf("[Name -%s- ID -%d-off line.]\n",msg.nick\_name,msg.pid);  
                Broadcast(h,msgid,&msg);  
                break;  
            default:  
                break;  
            }  
        }  
    }  
    else{//父进程用于服务器控制台  
        msg.mtype = TYPE\_SRV;   //因为不知道有多少个客户端，发给服务器后再转发  
        msg.subtype = 2;  
        while(1)  
        {  
            printf("[server console]\n");  
            fgets(msg.mtext,N,stdin);  
            //控制台收到quit时，通知客户端，server quit.  
            if(strncmp(msg.mtext,"quit",4) == 0){  
                msgsnd(msgid,&msg,MSG\_LEN,0);     
                sleep(2);  
                msgctl(msgid,IPC\_RMID,NULL);//删除消息队列  
                kill(pid,SIGKILL);  
                exit(0);  
            }  
        }  
    }  
    return 0;  
}  
void Broadcast(linklist h,int msgid,struct msgbuf \*msg)  
{  
    int i;  
    for(i = 0;i < Length(h);i++)  
    {  
        msg->mtype = GetValue(h,i);  
        if(msg->mtype != msg->pid){  
            msgsnd(msgid,msg,MSG\_LEN,0);  
        }         
    }  
}  
linklist CreateLinklist()  
{  
    linklist h;  
    h = (linklist)malloc(sizeof(listnode));  
    h->next = NULL;  
      
    return h;  
}  
//get the linklist's length  
int Length(linklist h)  
{  
    int n = 0;  
    h = h->next;  
    while(h)  
    {  
        n++;  
        h = h->next;  
    }  
    return n;  
}  
//judge if the linklist is empty  
int Empty(linklist h)  
{  
    return (NULL == h->next);  
}  
//insert the data whose value = 'value' at the end of the list  
int Insert(linklist h,int value)  
{  
    linklist p;  
    p = (linklist)malloc(sizeof(listnode));  
    p->data = value;  
    p->next = NULL;  
    while(h->next)  
    {  
        h = h->next;  
    }  
      
    h->next = p;  
    return 0;  
}  
//del a data by its value  
void Delete(linklist h,int value)  
{  
    linklist p = h->next;  
    //h points at the previous one,and p points at the later one  
    while(p)  
    {  
        if(p->data == value){  
            h->next = p->next;  
            free(p);  
            p = h->next;  
        }  
        else{  
            h = p;  
            p = h->next;  
        }  
    }  
    return;  
}  
//get the data at the position of pos  
int GetValue(linklist h,int pos)  
{  
    h = h->next;  
    while(pos--)  
    {  
        h = h->next;  
    }  
    return h->data;  
}

/\*  
\*用消息队列实现的简单聊天室程序(client)  
\*实现的功能：  
\*1。上线提醒  
\*2。下线提醒   
\*3。服务器断线后子进程都退出  
\* \*/  
#include <stdio.h>  
#include <sys/types.h>  
#include <sys/ipc.h>  
#include <sys/msg.h>  
#include <stdlib.h>  
#include <errno.h>  
#include <string.h>  
#include <unistd.h>  
#include <sys/wait.h>  
#include <signal.h>  
#define \_DEBUG\_   
#define TYPE\_SRV 10 //send 10是server的消息类  
#define N 128  
struct msgbuf{  
    long mtype;  
    int subtype;    //1->register 2->broadcast 3->unregister  
    int pid;  
    char nick\_name[N];  
    char mtext[N];  
};  
//定义消息正文大小的宏  
#define MSG\_LEN (sizeof(struct msgbuf) - sizeof(long))   
int main(int argc,char \*argv[])  
{  
    key\_t key;  
    int msgid;  
    pid\_t pid;  
    int TYPE\_ME;    //由getpid()来指定唯一该进程才有的mtype  
    struct msgbuf msg;   
    TYPE\_ME = getpid(); //用自己进程的进程号作为自己收消息的消息类型  
    //argv[1]用于接收昵称  
    if(argc < 2){  
        printf("Usage:%s <nick\_name>\n",argv[0]);  
        exit(-1);  
    }  
    //生成消息队列的key  
    if((key = ftok("./",'s')) < 0){  
        perror("ftok error.");  
        exit(-1);  
    }  
      
    //如果服务器已创建消息队列才可打开，客户端无权创建  
    if((msgid = msgget(key,0666)) < 0){  
        printf("[server maybe not on line...]\n");  
        exit(-1);  
    }  
    //每启动一个客户端要先实现一个注册操作  
    msg.mtype = TYPE\_SRV;     
       msg.subtype = 1;  
    msg.pid = getpid();  
    strcpy(msg.nick\_name,argv[1]);  
    msgsnd(msgid,&msg,MSG\_LEN,0);   //MSG\_LEN是消息正文大小   
          
    //客户端双进程实现读和写操作  
    if((pid = fork()) < 0){  
        perror("fork error.");  
        exit(-1);  
    }  
    else if(pid == 0){ //子进程专门负责读消息  
        while(1)  
        {  
            //msgid为消息队列的队列ID  
            //&msg为消息类型为TYPE\_ME的消息缓冲区首地址  
            //0指定了阻塞式等待  
            msgrcv(msgid,&msg,MSG\_LEN,TYPE\_ME,0);  
              
            //判断接收到的消息是否为quit  
            if(strncmp(msg.mtext,"quit",4) == 0){  
                printf("[server will close in 3 seconds...]\n");  
                kill(getppid(),SIGUSR1);//若收到quit通知父进程并自己退出  
                exit(0);  
            }  
            if(strncmp(msg.mtext,"exit",4) == 0){  
                msg.subtype = 3;  
            }  
                  
            //写成switch方便以后扩展  
            switch(msg.subtype)  
            {  
            case 1 :  
                printf("[%s on line]\n",msg.nick\_name);  
                break;  
            case 2 :  
                printf("[%s]\n",msg.nick\_name);  
                printf("%s",msg.mtext);  
                break;  
            case 3 :  
                printf("[%s off line]",msg.nick\_name);  
                break;  
            default :  
                break;  
            }  
        }  
    }  
    else{   //父进程专门负责写消息  
        msg.mtype = TYPE\_SRV;   //设定服务器的发送类型  
        msg.subtype = 2;    //设为广播模式  
        while(1)  
        {  
            printf("[%s]\n",msg.nick\_name);  
            fgets(msg.mtext,N,stdin);  
            msgsnd(msgid,&msg,MSG\_LEN,0);  
            //当此进程输入exit时，表示该进程退出聊天  
            //由服务器通知其他进程，该进程已下线      
            if(strncmp(msg.mtext,"exit",4) == 0){  
                msg.subtype = 3;  
                msgsnd(msgid,&msg,MSG\_LEN,0);  
                sleep(1);  
                kill(pid,SIGKILL);  
                exit(0);  
            }  
        }  
    }  
    return 0;  
}