1. 两个进程对一个共享内存各加1000万，通过pv操作，实现最终结果是2千万

#include <func.h>

#define NUM 10000000

int main()

{

    int shmid;

    shmid=shmget(1000,4096,IPC\_CREAT|0600);

    ERROR\_CHECK(shmid,-1,"shmget")

    printf("shmid=%d\n",shmid);

    int \*p=shmat(shmid,NULL,0);//共享内存连接到本进程内

    ERROR\_CHECK(p,(char\*)-1,"shmat")

    p[0]=0;

    int i;

    //信号量的创建及初始化

    int sem\_id=semget(1000,1,IPC\_CREAT|0600);//0600：给pv操作读写权限

    ERROR\_CHECK(sem\_id,-1,"semget")

    int ret=semctl(sem\_id,0,SETVAL,1);//给信号量的值初始化为1

    ERROR\_CHECK(ret,-1,"semctl")

    struct sembuf sopp,sopv;

    sopp.sem\_num=0;//操作的是几号信号量

    sopp.sem\_op=-1;//减1操作即p操作

    sopp.sem\_flg=SEM\_UNDO;

    sopv.sem\_num=0;

    sopv.sem\_op=1;//加1操作即v操作

    sopv.sem\_flg=SEM\_UNDO;

    //对内存操作产生了并发，并加锁，使结果等于20000000

    time\_t start,end;

    start=time(NULL);

    if(!fork())

    {

        for(i=0;i<NUM;i++)

        {

            //加锁，p操作，原子操作

            semop(sem\_id,&sopp,1);

            p[0]++;

            //解锁，v操作，原子操作

            semop(sem\_id,&sopv,1);

        }

        return 0;

    }else{

        for(i=0;i<NUM;i++)

        {

            //加锁，p操作，原子操作

            semop(sem\_id,&sopp,1);

            p[0]++;

            //加锁，v操作，原子操作

            semop(sem\_id,&sopv,1);

        }

        wait(NULL);

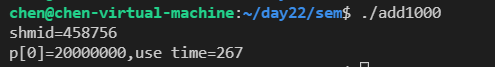
        end=time(NULL);

        printf("p[0]=%d,use time=%ld\n",p[0],end-start);

        return 0;

    }

}



1. 使用计数信号量，实现生产者消费者模式

#include <func.h>

//一次设置1个信号量集合中的多个信号量的值

//计数信号量实现生产者消费者模型

int main()

{

    //创建一个信号量集合

int sem\_id=semget(1000,2,IPC\_CREAT|0600);

ERROR\_CHECK(sem\_id,-1,"semget")

    unsigned short arr[2]={0,10};//0号代表产品个数，1号代表空位个数

    //给两个信号量同时赋值

    int ret=semctl(sem\_id,0,SETALL,arr);

    ERROR\_CHECK(ret,-1,"semctl")

    //父进程为生产者，子进程为消费者

    if(!fork())

    {//消费者

        struct sembuf sopp,sopv;

        sopp.sem\_num=0;//产品

        sopp.sem\_op=-1;

        sopp.sem\_flg=SEM\_UNDO;

        sopv.sem\_num=1;//空位

        sopv.sem\_op=1;

        sopv.sem\_flg=SEM\_UNDO;

        while(1)

        {

            printf("我是消费者,产品数目=%d,空位数=%d\n",semctl(sem\_id,0,GETVAL),semctl(sem\_id,1,GETVAL));

            semop(sem\_id,&sopp,1);

            printf("开始消费...\n");

            semop(sem\_id,&sopv,1);

            printf("我是消费者,产品数目=%d,空位数=%d\n",semctl(sem\_id,0,GETVAL),semctl(sem\_id,1,GETVAL));

            sleep(2);

        }

    }else{//生产者

        struct sembuf sopp,sopv;

        sopp.sem\_num=1;//空位

        sopp.sem\_op=-1;

        sopp.sem\_flg=SEM\_UNDO;

        sopv.sem\_num=0;//产品

        sopv.sem\_op=1;

        sopv.sem\_flg=SEM\_UNDO;

        while(1)

        {

            printf("我是生产者,产品数目=%d,空位数=%d\n",semctl(sem\_id,0,GETVAL),semctl(sem\_id,1,GETVAL));

            semop(sem\_id,&sopp,1);

            printf("开始生产...\n");

            semop(sem\_id,&sopv,1);

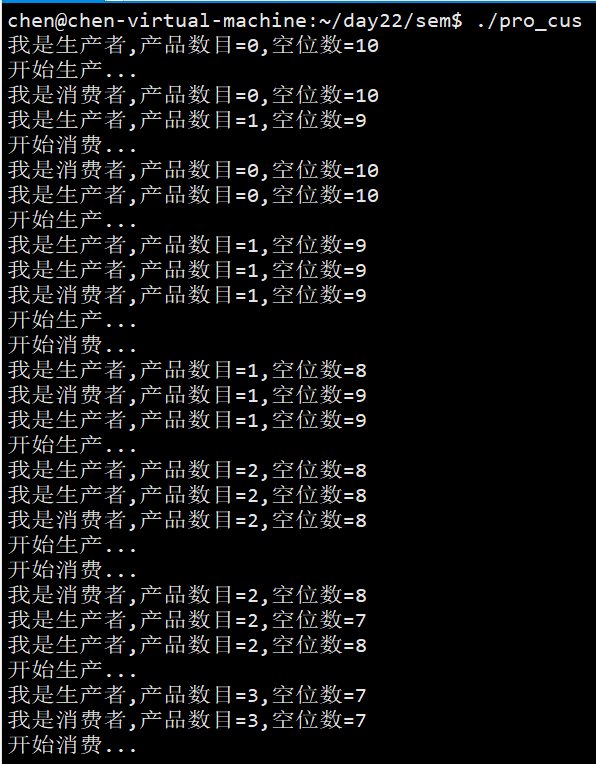
            printf("我是生产者,产品数目=%d,空位数=%d\n",semctl(sem\_id,0,GETVAL),semctl(sem\_id,1,GETVAL));

            sleep(1);

        }

    }

}



1. 创建消息队列，发送消息，接收消息，删除消息队列

//创建

#include <func.h>

int main()

{

    int msgid=msgget(1000,IPC\_CREAT|0600);//创建消息队列

    ERROR\_CHECK(msgid,-1,"mesgget")

    return 0;

}

//发送消息

#include <func.h>

struct MSG{

    long mtype;

    char buf[64];

};

int main()

{

    int msgid=msgget(1000,IPC\_CREAT|0600);//创建消息队列

    ERROR\_CHECK(msgid,-1,"mesgget")

    struct MSG msg;

    msg.mtype=1;

    strcpy(msg.buf,"hello");

    int ret=msgsnd(msgid,&msg,sizeof(msg.buf),0);//发送消息

    ERROR\_CHECK(ret,-1,"msgsnd")

    return 0;

}

//接收消息

#include <func.h>

struct MSG{

    long mtype;

    char buf[64];

};

int main()

{

    int msgid=msgget(1000,IPC\_CREAT|0600);//创建消息队列

    ERROR\_CHECK(msgid,-1,"mesgget")

    struct MSG msg;

    bzero(&msg,sizeof(msg));

    int ret=msgrcv(msgid,&msg,sizeof(msg),0,0);

    ERROR\_CHECK(ret,-1,"msgrcv")

    printf("get msg=%s\n",msg.buf);

    return 0;

}

//删除

#include <func.h>

int main()

{

    int msgid=msgget(1000,IPC\_CREAT|0600);//创建消息队列

    ERROR\_CHECK(msgid,-1,"mesgget")

    int ret=msgctl(msgid,IPC\_RMID,NULL);

    ERROR\_CHECK(ret,-1,"msgctl")

    return 0;

}

1. signal设置信号处理行为，演示不同信号到来时，后来的信号会打断原有信号的信号处理函数效果

#include <func.h>

void sigfunc(int signum)

{

    printf("before sleep,sig %d is coming\n",signum);

    sleep(3);

    printf("after sleep,sig %d is coming\n",signum);

}

int main()

{

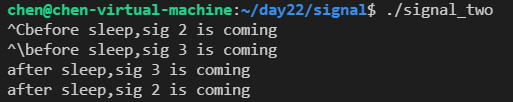
    signal(SIGINT,sigfunc);

    signal(SIGQUIT,sigfunc);

    while(1);

    return 0;

}



1. signal设置信号处理行为，演示相同信号到来时，无法打断本信号的效果

