1. **实验目的及内容**

通过编写多进程程序实现典型的生产者和消费者问题。

完成Windows版本和Linux版本。

一个大小为3的缓冲区，初始为空，2个生产者，随机等待一段时间，往缓冲区添加数据，若缓冲区已满，等待消费者取走数据后再添加，重复6次。

3个消费者，随机等待一段时间，从缓冲区读取数据，若缓冲区为空，等待生产者添加数据后再读取，重复4次，显示每次添加和读取数据的时间及缓冲区的状态。

**二、实验环境**

操作系统：windows10 64位

编译器：Visual Studio 2012

Linux Ubuntu 14.04.5

编译器：gcc

1. **基本原理及方法概述**

**2.1 Windows实现**

Windows利用共享主存及信号量实现，共三个.cpp文件，分别为main.cpp主要功能申请共享主存创建信号量以及生产者消费者进程。producer.cpp为生产者进程主要功能为在信号量的控制下向共享主存中放入信息。Consumer.cpp为消费者进程主要功能为在信号量的控制下从共享主存中取出信息。

使用CreateSemaphore()函数创建信号量。

使用OpenSemaphore()函数打开信号量获得句柄。

使用CreateFileMapping()函数创建共享文件句柄。

使用OpenFileMapping()函数打开共享文件

使用MapViewOfFile()函数在文件映射上创建视图。

使用CreateProcess()函数创建生产者消费者进程。

使用WaitForSingleObject()函数实现P操作。

使用ReleaseSemaphore()函数实现V操作。

**2.1.1主要系统调用函数说明**

**(1)创建信号量，信号量以名称为唯一标识**

HANDLE CreateSemaphore(

    LPSECURITY\_ATTRIBUTESlpSemaphoreAttributes, //信号量的安全属性。

    LONGlInitialCount, //信号量初值

    LONGlMaximumCount, //信号量最大值

    LPCTSTRlpName//信号量名称

);

**(2)实现P操作**

DWORD WaitForSingleObject(

HANDLE hHandle, //信号量的句柄

DWORD dwMilliseconds //INFINITE

);

**(3)实现V操作**

BOOL ReleaseSemaphore(

HANDLE hSemaphore, //要增加的信号量句柄

LONG lReleaseCount, //要增加的值

LPLONG lpPreviousCount //增加前的数值返回

);

1. **创建共享文件句柄，共享文件以名称为唯一标识**

HANDLE CreateFileMapping(  
  HANDLE hFile,        //物理文件句柄  
  LPSECURITY\_ATTRIBUTES lpAttributes, //安全设置  
  DWORD flProtect,     //保护设置  
  DWORD dwMaximumSizeHigh,  //高位文件大小  
  DWORD dwMaximumSizeLow,  //低位文件大小  
  LPCTSTR lpName      //共享内存名称  
);

**(5)打开共享文件**

HANDLE WINAPI OpenFileMapping(

DWORD   dwDesiredAccess, //FILE\_MAP\_WRITE

 BOOL    bInheritHandle, //TRUE

 LPCTSTR lpName //要打开的文件映射对象名称

);

**(6)在文件映射上创建视图**

LPVOID WINAPI MapViewOfFile(

HANDLE hFileMappingObject, //共享内存的句柄

 DWORD  dwDesiredAccess, // 可读写许可

 DWORD  dwFileOffsetHigh, //0

 DWORD  dwFileOffsetLow, //0

 SIZE\_T dwNumberOfBytesToMap //共享内存长度

);

**2.1.2主要的数据结构**

struct SHMbuff{

int stack;//生产者消费者的读写指针

int sum1;//记录生产者生产的产品数量

int sum2; // 记录消费者共取走多少产品

char buff[3][15];//大小为三的缓冲区

}; //共享主存的结构体

char szName[]="SHMbuff"; // 共享内存的名字

char NFULL[]="FULL"; //信号量FULL名字

char NEMPTY[]="EMPTY"; //信号量EMPTY名字

char NMUTEX[]="MUTEX"; //信号量MUTEX名字

**2.2 linux实现**

**2.2.1 系统调用函数说明**

**(1)创建共享内存区**

Shmget(key,size,shmflag);

key为内存区关键字，size是共享内存区字节长度，shmflag为创建或打开标志。

1. **将共享段附加到申请通信的进程空间**

shmat(shmid,shmadd,shmflg)

shmid为共享段标识，shmadd给出应附加到进程虚拟空间的地址，shmflg为允许对共享段的访问方式。

1. **将共享段与进程之间的链接解除**

shmdt(shmaddr)

shmaddr是共享段在进程地址空间的虚地址。

1. **创建信号量集合**

semget(key,nsems,semflg)

key为信号量集合的关键字，nsems为信号量集合中信号量的个数，semflg为创建或打开标志。

1. **对信号量执行控制操作**

semctl(semid,semnum,cmd,union semun arg)

semid为信号量集合的标识，semnum为信号量索引，cmd为要执行的操作命令，arg用于设置或返回信号量参数的信息。

用此函数可设置信号量的初值。

**2.2.2主要的数据结构**

struct SHMbuff{

int stack;//生产者消费者的读写指针

int sum1;//记录生产者生产的产品数量

int sum2; // 记录消费者共取走多少产品

char buff[3][15];//大小为三的缓冲区

}; //共享主存的结构体

void p(int semid,int sem\_num)//封装的P操作函数

{

struct sembuf xx;

xx.sem\_num=sem\_num;

xx.sem\_op=-1;

xx.sem\_flg=0;

semop(semid,&xx,1);

}

void v(int semid,int sem\_num)//封装的V操作函数

{

struct sembuf xx;

xx.sem\_num=sem\_num;

xx.sem\_op=1;

xx.sem\_flg=0;

semop(semid,&xx,1);

}

1. **程序流程图**

开始

**3.1 windows**

CreateSemaphore()创建三个信号量FULL,EMPTY,MUTEX

CreateFileMapping(),MapViewOfFile()创建共享文件及文件映射。

CreateProcess()创建两个生产者进程，每个进程生产六次

CreateProcess()创建三个消费者进程，每个进程消费四次

P(EMPTY)

P(MUTEX)

P(FULL)

P(MUTEX)

从缓冲区中取出产品

向缓冲区中放入产品

V(ENPTY)

V(MUTEX)

V(FULL)

V(MUTEX)

WaitForSingleObject()等待每个子进程结束

结束

**3.2 linux**

开始

Shmget()创建共享主存

Shmat()将共享段附加到申请通信的进程空间

Semget()创建信号量集合含有三个信号量，shmat()为信号量赋初值。共三个信号量FULL,EMPTY,MUTEX

fork()创建两个生产者进程，每个进程生产六次

fork创建三个消费者进程，每个进程消费四次

P(EMPTY)

P(MUTEX)

P(FULL)

P(MUTEX)

从缓冲区中取出产品

向缓冲区中放入产品

V(ENPTY)

V(MUTEX)

V(FULL)

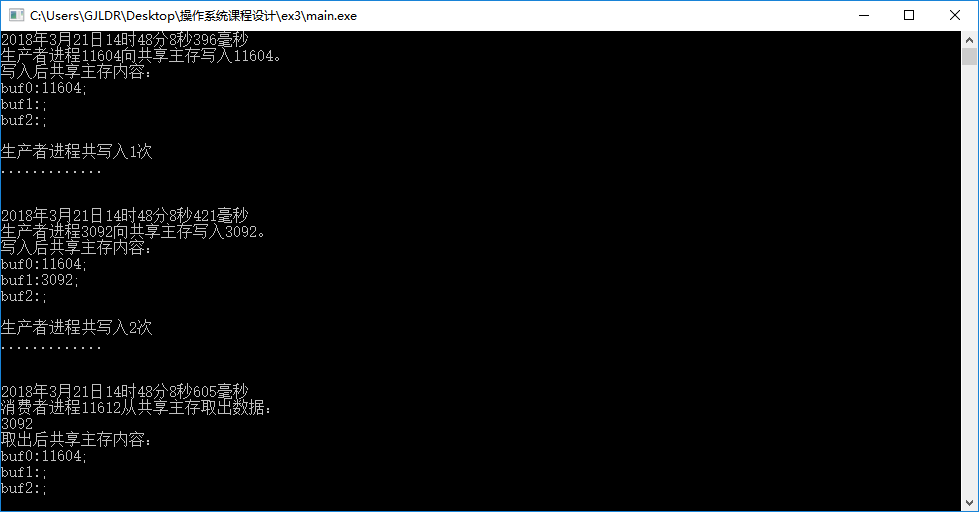
V(MUTEX)

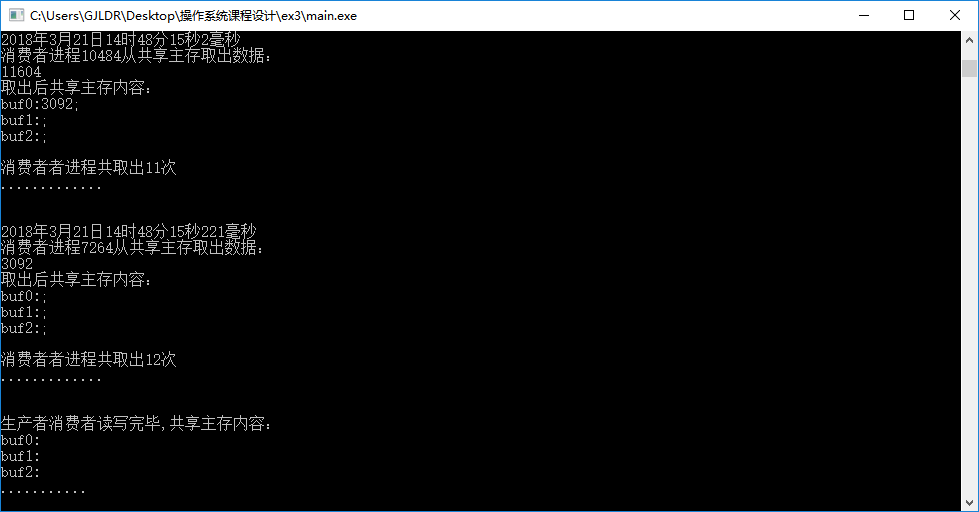
Wait()等待每个子进程结束

结束

1. **程序运行结果及分析**

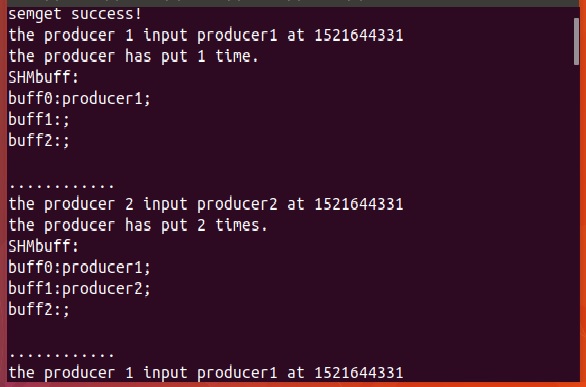
**4.1 windows**

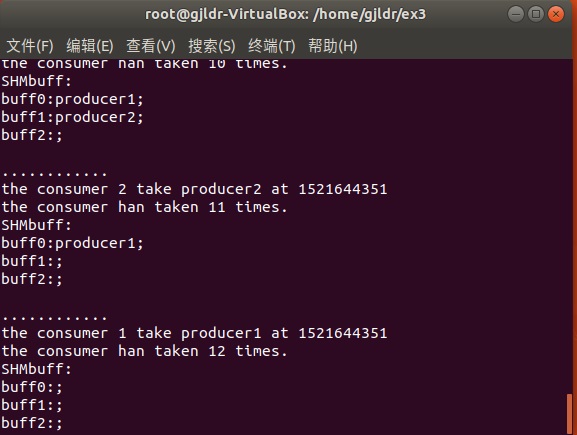




程序中两个生产者进程每个进程循环6次生产，共生产12个产品；三个消费者进程每个进程重复四次消费，共消费12次。完成后缓冲区为空。

**4.2 linux**

****

****

程序中两个生产者进程每个进程循环6次生产，共生产12个产品；三个消费者进程每个进程重复四次消费，共消费12次。完成后缓冲区为空。