****

实验三 生产者消费者

班 级： 07111505

姓 名： 徐宇恒

学 号： 1120151839

目录

[一. 实验目的 4](#_Toc512287913)

[二. 实验内容 4](#_Toc512287914)

[2.1. 创建两个生产者 4](#_Toc512287917)

[2.2. 创建三个消费者 4](#_Toc512287918)

[2.3. 创建一个大小为6的缓冲区 4](#_Toc512287919)

[2.4. 显示结果 4](#_Toc512287920)

[三. 实验环境 4](#_Toc512287921)

[四. 实验过程 5](#_Toc512287922)

[4.1. 基本思路 5](#_Toc512287925)

[4.1.1. “Program”程序： 5](#_Toc512287926)

[4.1.2. “Producer”程序： 5](#_Toc512287927)

[4.1.3. “Customer”程序： 5](#_Toc512287928)

[4.2. 伪代码 6](#_Toc512287929)

[4.3. 程序流程图 6](#_Toc512287930)

[4.4. Windows调用系统API 9](#_Toc512287931)

[4.4.1. sharemen 10](#_Toc512287932)

[4.4.2. Process\_INFORMATION 10](#_Toc512287933)

[4.4.3. CreateFileMapping 10](#_Toc512287934)

[4.4.4. MapViewOfFlie 10](#_Toc512287935)

[4.4.5. OpenFileMapping 10](#_Toc512287936)

[4.4.6. CreateSemaphore 11](#_Toc512287937)

[4.4.7. OpenSemaphore 11](#_Toc512287938)

[4.4.8. ReleaseSemaphore 11](#_Toc512287939)

[4.5. Linux调用系统API 11](#_Toc512287940)

[4.5.1. semget 11](#_Toc512287941)

[4.5.2. semctl 11](#_Toc512287942)

[4.5.3. semget 12](#_Toc512287943)

[4.5.4. semop 12](#_Toc512287944)

[4.5.5. shmat 13](#_Toc512287945)

[4.5.6. shmdt 13](#_Toc512287946)

[五. 实验结果 13](#_Toc512287947)

[5.1. Windows实验结果 13](#_Toc512287949)

[5.2. Linux实验结果 13](#_Toc512287950)

[六. 心得体会 14](#_Toc512287951)

# 实验目的

在Windows和Linux环境下利用进程模拟生产者消费者问题，通过自学互斥信号量来实现进程之间的互斥和同步操作，并采用共享内存来实现缓冲区的数据存储

# 实验内容



## 创建两个生产者

* 随机等待一段时间，向缓冲池中添加数据。
* 若缓冲区已满，等待消费者取走数据后在添加
* 重复12次

## 创建三个消费者

* 速记等待一段时间，从缓冲区中取出数据。
* 若缓冲区为空，则等待生产者添加数据
* 重复8次

## 创建一个大小为6的缓冲区

* 初始为空
* 允许消费者进程读数据
* 允许生产者进程写数据

## 显示结果

* 显示每次添加和读取数据的时间
* 显示缓冲区的状态

# 实验环境

|  |  |  |
| --- | --- | --- |
|  | Windows | Linux |
| 操作系统 | Windows10 Pro 64bit | Ubuntu 16.04LTS |
| 编译器 | Visual Studio 2017 IDE | Gcc |

# 实验过程



## 基本思路

在Windows和Linux下分别编写三个程序，主程序命名为“Program”，两个子程序“Producer”，“Customer”。

### “Program”程序：

* 建立并初始化缓冲区
* 创建互斥信号量Mutex，空信号量Empty，满信号量Full
* 创建生产者和消费者子进程

### “Producer”程序：

* 打开缓冲区
* 打开互斥信号量、空信号量、满信号量
* 对空信号量和互斥信号量进行P操作
* 申请成功则放入数据
* 对满信号量和互斥信号量进行V操作
* 返回到第三步，重复12次

### “Customer”程序：

* 打开缓冲区
* 打开互斥访问信号量、空信号量、满信号量
* 若打开成功，对满信号量做P操作，对互斥信号量做P操作
* 若申请成功，则读取数据，并将当前缓冲区置零
* 对空信号量做V操作，对互斥信号量做V操作。
* 返回第三步，重复8次。

## 伪代码

|  |
| --- |
| semaphore Mutex = 1;  semaphore Fill = 0;  semaphore Empty = 3;  procedure producer()  {  while (i<6)  {  P(Empty);  P(Mutex);  PutItemIntoBuffer(item);  V(Full);  V(Mutex);  }  }  procedure consumer()  {  while (i<4)  {  P(Full);  P(Mutex);  RemoveItemFromBuffer();  V(Empty);  V(mutex);  }  } |

## 程序流程图

主控程序开始

建立缓冲区

创建互斥、空、满信号量

创建成功？

等待

创建消费者进程

返回错误信息

创建生产者进程

主控程序结束

结束？

 是

 否

生产者程序开始

打开缓冲区

打开成功

打开互斥、满、空信号量

返回错误信息

i == 0

i<12?

填写缓冲区

P(Mutex)

P(Empty)

打印当前状态

V(Full)

V(Mutex)

i++

结束

 是

 是

 否

 否

生产者程序开始

打开缓冲区

打开成功

打开互斥、满、空信号量

返回错误信息

i == 0

i<8?

读取缓冲区

P(Mutex)

P(Full)

打印当前状态

V(Empty)

V(Mutex)

i++

结束

 是

 是

 否

 否

## Windows调用系统API

### sharemen

### Process\_INFORMATION

### CreateFileMapping

|  |
| --- |
| 创建新的文件映射内核对象  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  ); |

### MapViewOfFlie

|  |
| --- |
| 将一个文件映射对象映射到当前应用程序的地址空间。  LPVOID MapViewOfFile(  HANDLE hFileMappingObject, //共享文件对象  DWORD dwDesiredAccess, //文件共享属性  DWORD dwFileOffsetHigh, //文件共享区的偏移地址的高32位  DWORD dwFileOffsetLow, //文件共享区的偏移地址的低32位  SIZE\_T dwNumberOfBytesToMap //映射文件的字节数  ); |

### OpenFileMapping

|  |
| --- |
| HANDLE OpenFileMapping(  DWORD dwDesiredAccess, //映射对象的文件数据的访问方式  BOOL bInheritHandle, //返回句柄能否由当前进程启动的新进程继承  LPCSTR lpName //要打开的文件映射对象名称  ); |

### CreateSemaphore

|  |
| --- |
| 创建一个新的信号量。  HANDLE WINAPI CreateSemaphore(  LPSECURITY\_ATTRIBUTES lpSemaphoreAttributes, //安全属性  LONG lInitialCount, //信号量的初始值  LONG lMaximumCount, //信号量的最大值  LPCTSTR lpName //信号量的名称  ); |

### OpenSemaphore

|  |
| --- |
| 为现有的一个已命名信号量对象创建一个新句柄。  HANDLE WINAPI OpenSemaphore(  DWORD dwDesiredAccess, //指定访问权限  BOOL bInheritHandle, //是否允许子进程继承句柄  LPCTSTR lpName //要打开的对象的名字  ); |

### ReleaseSemaphore

|  |
| --- |
| 用于对指定的信号量增加指定的值。  BOOL WINAPI ReleaseSemaphore(  HANDLE hSemaphore, //所要操作的信号量对象的句柄  LONG lReleaseCount, //这个信号量对象在当前基础上所要增加的值  LPLONG lpPreviousCount //指向返回信号量上次值的变量的指针  ); |

## Linux调用系统API

### semget

|  |
| --- |
| 得到一个信号量集标识符或创建一个信号量集对象并返回信号量集标识符  int semget(  key\_t key, //所创建或打开信号量集的键值。  int nsems, //创建的信号量集中的信号量的个数  int semflg //调用函数的操作类型  ); |

### semctl

|  |
| --- |
| 得到一个信号量集标识符或创建一个信号量集对象并返回信号量集标识符。  int semctl(  int semid, //要操作的信号量  int semnum, //信号量编号  int cmd, //指定对信号量的操作  union semun arg //  ); |

### semget

|  |
| --- |
| 得到一个共享内存标识符或创建一个共享内存对象并返回共享内存标识符  int shmget(  key\_t key, //标识符的规则  size\_t size, //共享存储段的字节数  int shmflg //读写的权限  ); |

### semop

|  |
| --- |
| 操作一个或一组信号。  int semop(  int semid, //信号量集标识符  struct sembuf \*sops, //指向进行操作的信号量集结构体数组的首地址  unsigned nsops //进行操作信号量的个数  );  struct sembuf {  short semnum;  /\*信号量集合中的信号量编号，0代表第1个信号量\*/  short val;  /\*若val>0进行V操作信号量值加val，表示进程释放控制的资源 \*/  /\*若val<0进行P操作信号量值减val，若(semval-val)<0（semval为该信号量值），则调用进程阻塞，直到资源可用；若设置IPC\_NOWAIT不会睡眠，进程直接返回EAGAIN错误\*/  /\*若val==0时阻塞等待信号量为0，调用进程进入睡眠状态，直到信号值为0；若设置IPC\_NOWAIT，进程不会睡眠，直接返回EAGAIN错误\*/  short flag;  /\*0 设置信号量的默认操作\*/  /\*IPC\_NOWAIT设置信号量操作不等待\*/  /\*SEM\_UNDO 选项会让内核记录一个与调用进程相关的UNDO记录，如果该进程崩溃，则根据这个进程的UNDO记录自动恢复相应信号量的计数值\*/  }; |

### shmat

|  |
| --- |
| 连接共享内存标识符为shmid的共享内存，连接成功后把共享内存区对象映射到调用进程的地址空间，随后可像本地空间一样访问.  void \*shmat(  int shmid, //共享内存标识符  const void \*shmaddr, //指定共享内存出现在进程内存地址的什么位置  int shmflg //SHM\_RDONLY：为只读模式，其他为读写模式  ); |

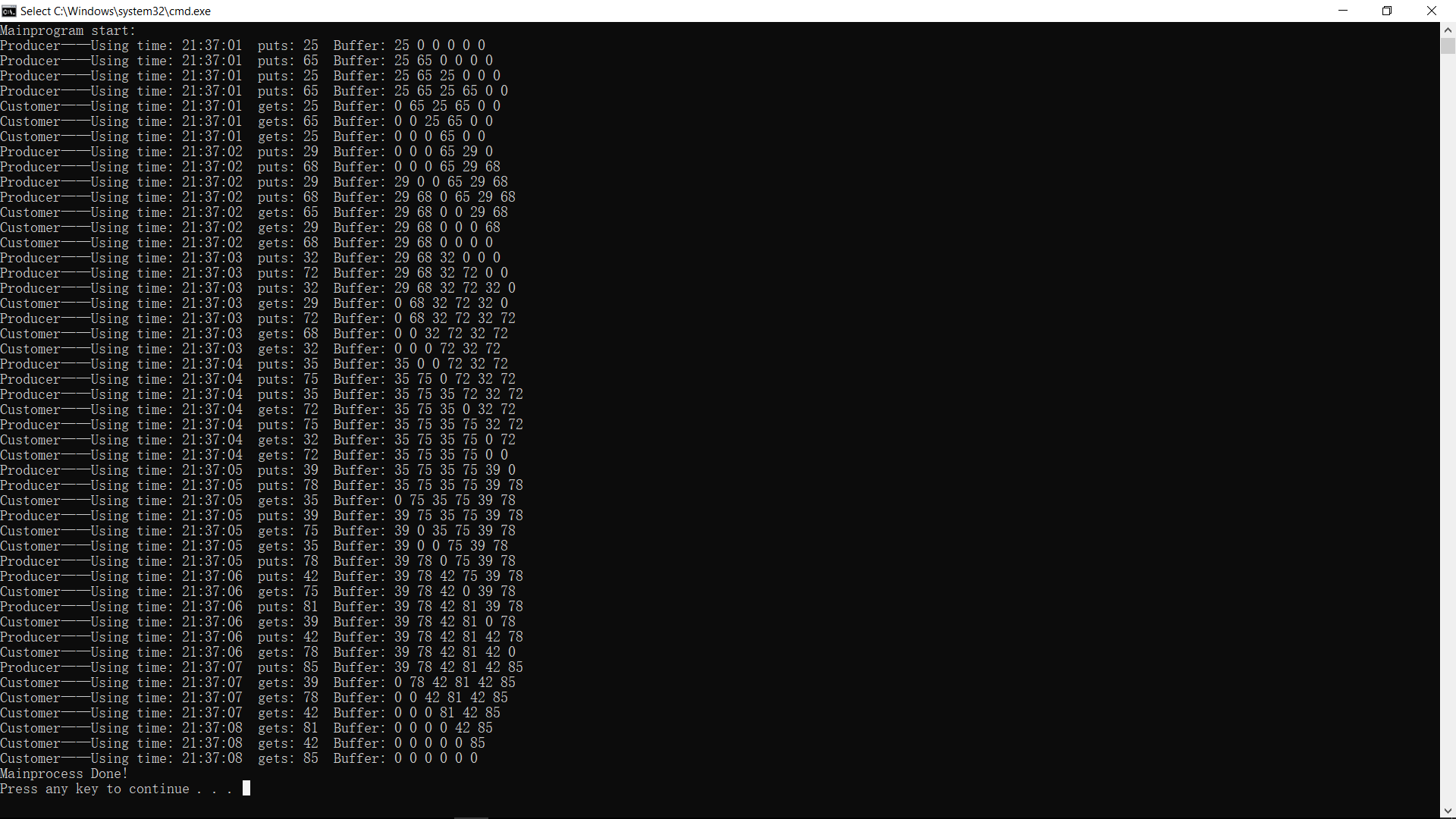
### shmdt

|  |
| --- |
| 与shmat函数相反，是用来断开与共享内存附加点的地址，禁止本进程访问此片共享内存  int shmdt(  const void \*shmaddr //连接的共享内存的起始地址  ) |

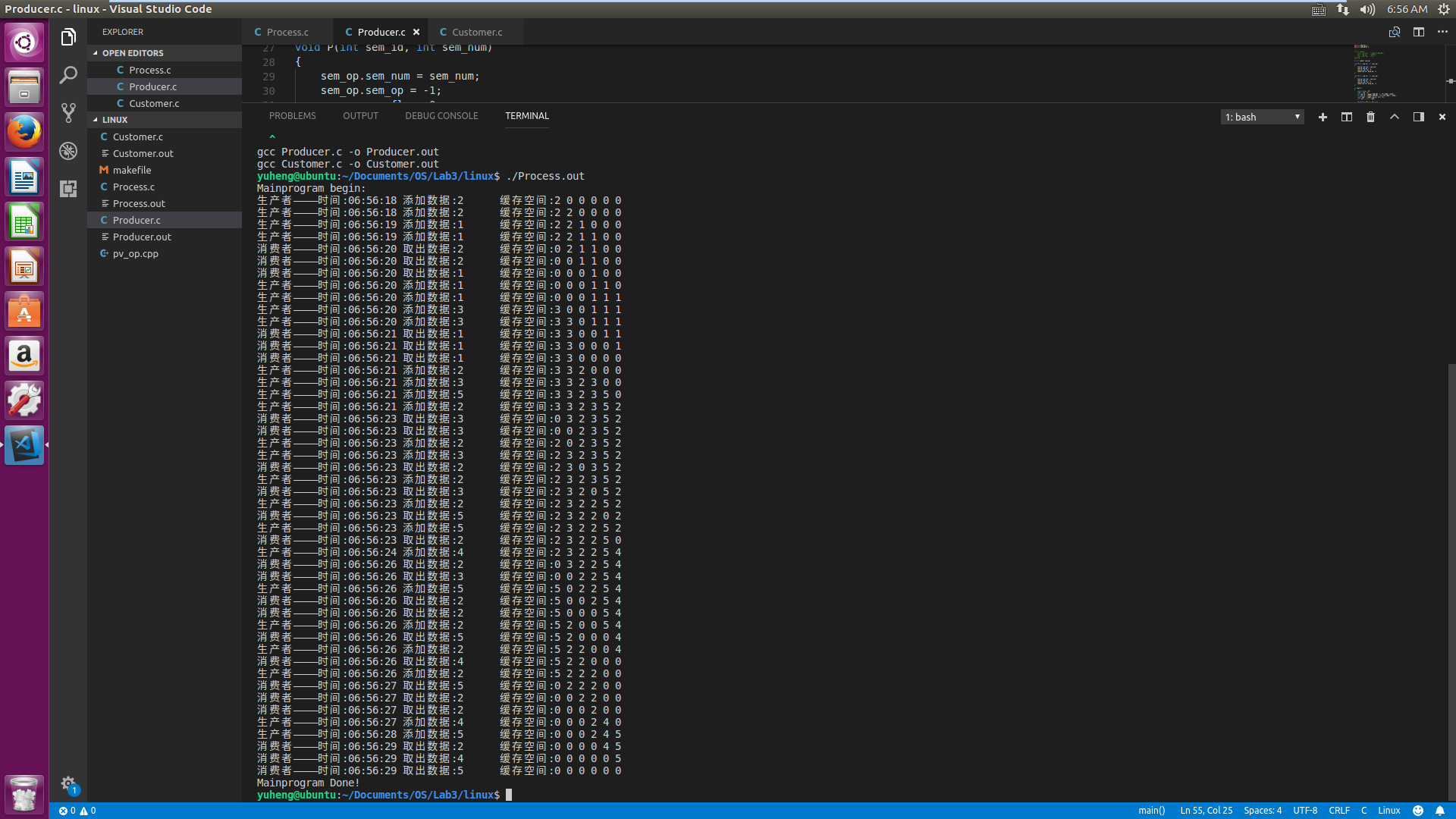
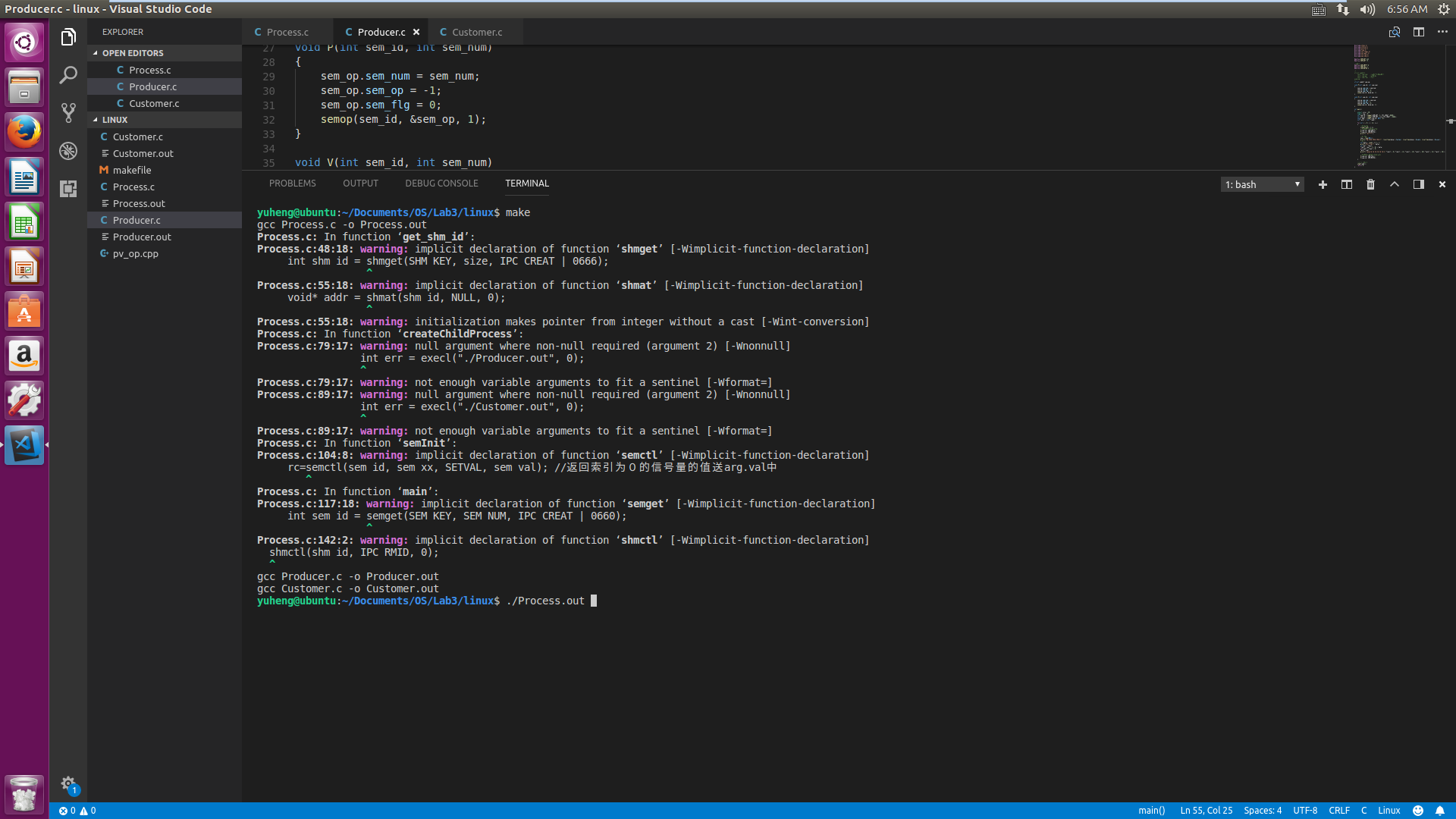
# 实验结果



## Windows实验结果



## Linux实验结果



# 心得体会

通过本次实验，基掌握了通过本次实验，基掌握了Windows和Linux中通过互斥体和信号量实现控制进程同步和互斥的方法，并初步学习并实现了共享主存，为多进程通信和共享数据提供了更加快捷方便的方法。

体会1：在子进程中，应当先申请信号量，再申请互斥体，顺序不能改变。否则会发生死锁。

体会2：在申请信号量和互斥体成功之前，子程序并未开始真正的执行，第一次写程序时，将printf (“生产者——”);放在了申请之前，造成输出结果混乱。

体会3：加深了int型数据占四个字节的知识点，第一次写程序时忘了每次读取一个新的int型数据时，地址需要加4，导致出现错误。