操作系统课程设计实验报告

**实验三 生产者与消费者进程模拟**

指导教师：陆慧梅老师

班 级：07111507

学 号：1120151880

姓 名：廖汉龙

邮 箱：[liaohanlong@outlook.com](mailto:liaohanlong@outlook.com)

2018 年 3 月 20 日

**一、实验目的**

1. 掌握Windows、Linux下利用信号量机制实现进程的同步，实践操作系统中经典的生产者与消费者的问题。

2. 掌握信号量、互斥量、缓冲区的建立及相关操作。

3. 掌握Windows、Linux不同环境下实现内存共享的方法，创建和使用共享内存区，使多个进程可以使用同一块数据。

**二、实验内容**

**1. 总体要求：**

在Windows和Linux系统下通过进程编程模拟生产者消费者算法。

设计一个大小为3的缓冲区。

**2. 生产者要求：**

创建2个生产者；

每个生产者随机等待一段时间，向缓冲区添加一个大写字母，若缓冲区已满，等待消费者取走字母后再添加；

重复6次；

**3. 消费者要求：**

创建3个消费者；

每个消费者随机等待一段时间，从缓冲区读取字母；

若缓冲区为空，等待生产者添加字母后再读取；

重复4次；

**4. 打印内容要求；**

生产者打印：生产者本次写入缓冲区字母；

消费者打印：消费者本次取走的字母；

需打印缓冲区内容；

按先生产的商品先消费原则；

**三、实验环境及配置方法**

1. **Windows环境：**

操作系统：Windows 10 家庭中文版

编译环境：Visual Studio Code

gcc 4.9.2

1. **Linux环境：**

操作系统：Ubuntu 16.04 LTS

编译环境：GCC 5.4.0

编写代码软件：Visual Studio Code

**四、实验方法与步骤**

**1.实验原理**

生产者进程与消费者进程存在同步与互斥关系：进程互斥地访问缓冲区，对缓冲区内容进行读写操作；而各个进程之间是同步运行的关系。根据操作系统课程的学习内容，需要设定以下信号量进行进程间的操作：

MUTEX：互斥信号量，控制生产者进程与消费者进程互斥的访问缓冲区，初始值为1；

EMPTY：表示缓冲区中空闲区的大小，初始值为3；

FULL：表示缓冲区中非空闲区的大小，初始值为0

伪代码如下：

|  |
| --- |
|  |

各个过程的流程图如下：

|  |
| --- |
|  |

图-主控过程

|  |
| --- |
|  |

图-生产者过程

|  |
| --- |
|  |

图-消费者过程

**2.window 系统与Linux 下的API介绍**

**（1）windows 系统下的API**

* CreateFileMapping()，打开或创建一个文件映射对象。

|  |
| --- |
|  |

若函数调用成功，返回新建文件映射对象句柄。如果函数调用失败，返回值为0，并且设置错误信息，错误信息可以通过GetLastError返回错误原因。即使函数成功，但倘若返回的句柄属于一个现成的文件映射对象，那么GetLastError也会设置成ERROTR\_ALREADY\_EXISTS.在这种情况下，文件的映射长度就是先有对象的长度，而不是这个函数指定的尺寸。

* OpenFileMapping（）打开一个已经存在的文件映射对象

|  |
| --- |
|  |

如果函数调用成功，则返回要打开的文件映射对象的句柄；否则返回值为0，则设置错误信息，并通过GetLastError返回错误原因。

* MapViewofFile() 将文件对象映射到进程的地址空间

|  |
| --- |
|  |

如果调用成功，返回文件映射在内存中的其实地址，否则返回值为0。且会设置错误信息，并通过GetLastError返回错误原因，MapViewofFileEX()函数允许指定一个基本地址来进行映射。

* UnmapView() 解除进程对于一个文件映射对象的映射

|  |
| --- |
|  |

如果函数调用成功，返回值为非0，否则返回值为0，切会设置错误信息，并通过GetlastError返回错误信息

* createSemphore() 创建一个信号量对象

|  |
| --- |
|  |

如果函数调用成功，则会返回一个信号量对象的句柄，否则返回NULL.不论是哪种情况，GetLastError()函数都会返回一个合理的结果。

* ReleaseSemaphore() 释放信号量

|  |
| --- |
|  |

如果函数成功调用，则返回true, 否则返回false，失败时可以通过调用GetLastError（）函数获得原因。

* OpenSemaphore() 打开一个信号量

|  |
| --- |
|  |

如果调用成功，返回信号量对象的句柄，否则，返回为NULL。一旦不再需要，一定要使用CloseHandle关闭信号量句柄。如果对象的所有句柄都已经关闭那么该对象就会被删除。

**（2）Linux 系统的API**

* semget() 创建一个信号量集合

|  |
| --- |
|  |

返回值为信号量集合的标志号，出错返回-1；

* semop() 对信号量进程P/V操作

|  |
| --- |
|  |

其操作命令由用户提供的信号量操作模板（sembuf）定义，改模板的结构如下

|  |
| --- |
|  |

正常返回值为0，错误返回值为-1

* semctl() 对信号量执行控制操作

|  |
| --- |
|  |

其中，semun定义如下：

|  |
| --- |
|  |

* shmget() 申请一个共享内存区

|  |
| --- |
|  |

成功时，为共享内存区域标识区的标识，不成功返回-1，errno存储错误原因

* shmat() 将共享段附加到申请通信的进程空间

|  |
| --- |
|  |

* shmdt() 将共享段与进程之间接触连接

|  |
| --- |
|  |

函数的返回值为0

* shmctl() 对共享内存区执行控制操作

|  |
| --- |
|  |

若成功，返回0，失败返回-1.

**3.代码实现过程**

**(1)windows 环境**

**1). 初始化缓冲区**

定义缓冲区结构，采用循环队列实现先生产的商品先消费的原则。

结构体定义代码截图：

|  |
| --- |
|  |

在此处，我没有像网上给的写法，把信号量和共享内存的区域分开定义，而是统一定义在了一个结构体里边，因为信号量也是所有进程共享的资源，这样构造，使得后续的代码简洁许多。

对缓冲区进行初始化，创建共享内存区域，使用函数CreateFileMapping()来创建一个共享的文件数据句柄，使用MapViewOfFile()来获取共享的内存地址

|  |
| --- |
|  |

对信号量的初始化在主进程中实现，当当前为父进程的时候，则会对信号量进程初始化。

分别创建互斥信号量Mutex，表示缓冲区空闲区数量的信号量Empty，表示缓冲区非空闲区的信号量Full。

|  |
| --- |
|  |

**2) 创建子进程**

通过自定义一个StartClone函数，创建子进程

|  |
| --- |
|  |

这个过程在父进程中使用一个循环实现一次性创建所有的子进程，包括生产者与消费者

|  |
| --- |
|  |

**3) 生产者进程实现**

生产者进程的主要工作流程：

申请一个空缓冲区->申请对于缓冲区的操作->在缓冲区中写入字母->释放缓冲区->增加缓冲区中非空闲区的数量。

在这核心操作的过程前后，关键是对信号量的操作，可以发现，在网上与学长的一些代码中非常奇怪的并没有考虑到互斥信号量Mutex的作用，经过实验发现，没有考虑互斥信号量在windows系统的小数量级别的实验中不会出现问题，但是在大数量级别的时候或者在Linux系统下，则会出现明显的错误。

由于在开始时候的缓冲区的重新定义，所以此处对于缓冲区的操作则会简单许多。此处关键代码截图如下：

|  |
| --- |
|  |

|  |
| --- |
|  |

**4) 消费者进程实现**

消费者的进程实现基本和生产者相似，代码可以直接拷贝修改，除了对于信号量的操作意外，就是对于缓冲区的操作，操作大致如下：

消费者进程的主要工作流程：申请一个缓冲区的非空闲区->申请对于缓冲区的操作->从缓冲区中读取字母->释放缓冲区->增加一个缓冲区中空闲区的数量。

关键代码截图如下：

|  |
| --- |
|  |

**6) 输出函数**

为了对于缓冲区以及当前的进程的变化的过程有更加直观的认识，定义了一个输出函数。

|  |
| --- |
|  |

**7) 编译运行代码，观察实验结果。**

**(2)Linux 系统环境**

**1)初始化缓冲区**

定义缓冲区结构，采用循环队列实现先生产的商品先消费的原则。

结构体定义代码截图：

|  |
| --- |
|  |

初始化共享内存区域，利用函数shmget()申请共享区域，使用函数shmat()得到映射到共享区域的指针，并对缓冲区进行初始化。代码截图如下：

|  |
| --- |
|  |

**2) 初始化信号量**

利用函数semget()创建信号量集合，利用函数semctl()分别创建互斥信号量mutex、空信号量empty、满信号量full。

|  |
| --- |
|  |

**3) 定义P操作与V操作**

P操作使信号量减一，代码截图如下：

|  |
| --- |
|  |

V操作使信号量加一，代码截图如下：

|  |
| --- |
|  |

**4)创建子进程**

创建生产者并且进程对于缓冲区的操作如下：

生产者进程的主要工作流程：申请一个空缓冲区->申请对于缓冲区的操作->在缓冲区中写入字母->释放缓冲区->增加缓冲区中非空闲区的数量。

与Window操作系统下一致的是对于信号量的操作，关键代码截图如下：

|  |
| --- |
|  |

**5)消费者进程实现**

消费者进程的主要工作流程：申请一个缓冲区的非空闲区->申请对于缓冲区的操作->从缓冲区中读取字母->释放缓冲区->增加一个缓冲区中空闲区的数量。关键代码截图如下：

|  |
| --- |
|  |

**6)自定义输出函数**

为了与Windows系统下的输出保持一致，我同样在Linux下定义了相同的输出方式函数：

|  |
| --- |
|  |

**7)编译运行代码，观察实验结果。**

**五、实验结果和分析**

**1.Windows环境**

程序运行结果截图：

|  |
| --- |
|  |

|  |
| --- |
|  |

结果分析：

由图可见，缓冲区初始化为空，按照先生产先消费的顺序，生产者进程向缓冲区中写入字母，消费者进程从缓冲区中读取字母；缓冲区存满之后，生产者无法继续写入，只有等待消费者读取之后才可以继续写入。在进程运行的整个过程中，生产者共有2个，每一个生产者随机进行6次写字母操作；消费者有3个，每一个消费者随机进行4次读字母操作。共有二十四次进程操作，缓冲区最后的内容为空。

实现了生产者进程与消费者对于缓冲区的互斥操作，与预期结果一致。

1. Linux环境

|  |
| --- |
|  |

|  |
| --- |
|  |

结果分析：

由图可见，运行结果与Windows环境下结果相同，说明在Linux系统下利用信号量机制实现了生产者进程与消费者进程对于缓冲区的互斥读写操作，与预期结果一致。

**六、讨论、心得**

本次实验内容利用信号量机制，实现进程之间的同步和互斥关系。

对于信号量机制，在之前的操作系统课程中，有了一定的理论上的了解，但对于代码实现，确实有很多不懂得地方，这个实验确实比较难。我通过阅读课本上的函数示例与网上大量的参考资料，也参考了一些学长的代码，也发现了一些存在的问题。

另外，通过这次实验，我还学会了共享内存的使用，使得不同的进程可以访问一块相同的内存区域，来进行数据交换，提高了效率，但为了正确使用共享内存，就需要使用信号量机制对其进行互斥与同步关系控制。

另外还有一个问题是始终还没有解决的，在小规模的数据量下，windows系统下的随机性并没有Linux表现好，总是出现生产者与消费者交替访问缓冲区的情况，在规模较大的时候，随机性较好。