

实验三 生产者与消费者问题

班级：07121502 学号：1320151098 姓名：赵璐

1.实验目的：

在 Windows 和 Linux 环境下利用进程模拟生产者和消费者问题，通过自学互斥体和信号量来实现进程之间的同步和互斥，并采用共享主存来实现缓冲区的数
据存取。

2.实验内容：

1. 创建 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

2. 创建 3 个消费者

随机等待一段时间，往缓冲区读取数据，

若缓冲区已空，等待生产者添加数据后再读取

重复 4 次

3. 创建一个大小为 3 的缓冲区，初始为空

4. 结果显示每次添加和读取数据的时间及缓冲区的状态

5. 完成 Windows 和 Linux 版本

3.实验环境：

Windows 下的 Visual Studio 2015；Linux 下的 Gcc；

4.实验过程：

4.1 基本思路

在 Windows 和 Linux 下分别编写三个程序，主程序命名为“Program”，被调用

的两个子程序命名为“Producer”、“Customer”。

“Program”程序：

- a. 建立并初始化缓冲区
- b. 创建互斥访问信号量 Mutex、空信号量 Empty、满信号量 Full
- c. 若创建成功，则创建生产者子进程和消费者子进程
- d. 运行结束

“Producer”程序：

- a. 打开缓冲区
- b. 打开互斥访问信号量、空信号量、满信号量
- c. 若打开成功，对空信号量做 P 操作，对互斥信号量做 P 操作
- d. 若申请成功，则放入数据
- e. 对满信号量做 V 操作，对互斥信号量做 V 操作。
- f. 程序结束

步骤 c、d、e 循环 6 次

“Customer”程序：

- a. 打开缓冲区
- b. 打开互斥访问信号量、空信号量、满信号量
- c. 若打开成功，对满信号量做 P 操作，对互斥信号量做 P 操作
- d. 若申请成功，则读取数据，并将当前缓冲区置零
- e. 对空信号量做 V 操作，对互斥信号量做 V 操作。
- f. 程序结束

步骤 c、d、e 循环 4 次

4.2 数据结构及函数说明

(1)共享主存区结构

```
struct sharemen
{
    int put;           //记录放数据位置
    int take;          //记录取数据位置
    int number[3];     //数据缓存去
};
```

(2)创建进程 CreateProcess()函数

```
//返回值：创建成功返回TRUE，否则返回FALSE
Bool CreateProcess(
    //指定了新进程将使用的可执行文件和传递给新进程的命令行字符串
    LPCTSTR lpszApplicationName,
    LPCTSTR lpCommandLine,
    //指向进程和线程安全属性结构的指针
    LPSECURITY_ATTRIBUTES lpProcessAttributes,
    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    //新进程是否继承调用进程的打开句柄的副本
    BOOL bInheritHandles,
    //包含几个标志组合
    DWORD dwCreationFlags,
    //指向新进程的环境块，若为空，则使用父进程的环境块
    LPVOID lpEnvironment,
    //子进程的工作路径
    LPCTSTR lpCurrentDirectory,
    //指向STARTUPINFO结构体，得到启动信息
    LPSTARTUPINFO lpStartupInfo,
    //指向PROCESS_INFORMATION结构体，返回进程和线程信息
    LPPROCESS_INFORMATION lpProcessInformation
);
```

该函数可使系统创建一个进程内核对象和一个线程内核对象。且打开进程和线程对象，并将与进程相关的每个对象句柄放入PROCESS_INFORMATION的结构中。

(3)PROCESS_INFORMATION 结构

包含新创建进程的信息

```
typedef struct _PROCESS_INFORMATION{
    //新创建进程的句柄
    HANDLE hProcess;
    //新创建进程的主线程的句柄
    HANDLE hThread;
    //新创建进程的标识
    DWORD dwProcessId;
    //新创建进程的主线程的标识
    DWORD dwThreadId;
} PROCESS_INFORMATION, *LPPROCESS_INFORMATION;
```

(4)关闭对象函数 CloseHandle()函数

BOOL CloseHandle(HANDLE hObject);

参数：hObject 代表一个已打开对象的句柄。

(5)WaitForSingleObject()函数

```
DWORD WaitForSingleObject(  
    //被等待对象的句柄  
    HANDLE hObject,  
    //指定等待时间, INFINITE代表一直等待  
    DWORD dwMilliseconds  
);
```

(6)fork()函数

pid_t fork(void);

fork()函数调用后，子进程是父进程的一个拷贝，也即子进程也有了一个独立的执行体。

返回值：正确完成时，返回给父进程的是被创建子进程的标识，返回给子进程自己的是 0；创建失败时，返回父进程的是-1.

(7)execv()函数

```
int execv(  
    //可执行文件的路径名  
    const char *pathname ;  
    //文件名  
    char *const arg[]  
)
```

execv()函数来运行一个新程序，然后该进程的代码段、数据段和堆栈段就被新程序所代替。

(8)wait()函数

*pid_t wait(int *statloc);*


statloc 是一个整型指针，指出存放子进程的终止状态码的位置。若不关

心子进程的终止状态，可传递一个空指针。父进程通过调用 wait()函数同步等待子进程结束，并在得到终止状态码后立即返回。

5.实验结果分析

5.1Windows 下的结果

1. 首先运行两个子程序 WCustomer.cpp 和 WProducer.cpp, 使其生成.exe 可执行文件，并将其放入指定目录下。
2. 在 VS 中直接运行 WProgram.cpp。
3. 结果显示如下：



```
E:\编程代码\PandC\Debug\PandC.exe
Mainprogram Start:
生产者--时间 13:18:20 放入数据:51 缓存状态:51 0 0
生产者--时间 13:18:20 放入数据:78 缓存状态:51 78 0
生产者--时间 13:18:20 放入数据:51 缓存状态:51 78 51
消费者--时间 13:18:21 取出数据:51 缓存状态:0 78 51
消费者--时间 13:18:21 取出数据:78 缓存状态:0 0 51
消费者--时间 13:18:21 取出数据:51 缓存状态:0 0 0
生产者--时间 13:18:21 放入数据:81 缓存状态:81 0 0
生产者--时间 13:18:21 放入数据:54 缓存状态:81 54 0
消费者--时间 13:18:21 取出数据:81 缓存状态:0 54 0
生产者--时间 13:18:21 放入数据:54 缓存状态:0 54 54
消费者--时间 13:18:21 取出数据:54 缓存状态:0 0 54
消费者--时间 13:18:21 取出数据:54 缓存状态:0 0 0
生产者--时间 13:18:21 放入数据:81 缓存状态:81 0 0
消费者--时间 13:18:21 取出数据:81 缓存状态:0 0 0
生产者--时间 13:18:21 放入数据:54 缓存状态:0 54 0
消费者--时间 13:18:21 取出数据:54 缓存状态:0 0 0
生产者--时间 13:18:21 放入数据:54 缓存状态:0 0 54
生产者--时间 13:18:21 放入数据:81 缓存状态:81 0 54
消费者--时间 13:18:21 取出数据:54 缓存状态:81 0 0
消费者--时间 13:18:21 取出数据:81 缓存状态:0 0 0
生产者--时间 13:18:21 放入数据:81 缓存状态:0 81 0
消费者--时间 13:18:21 取出数据:81 缓存状态:0 0 0
生产者--时间 13:18:21 放入数据:81 缓存状态:0 0 81
消费者--时间 13:18:21 取出数据:81 缓存状态:0 0 0
Mainprogram Done!
```

5.2Linux 下的结果

1. 打开终端，进入到有主、子程序的目录中。首先生产两个子程序 LCustomer.cpp 和 LProducer.cpp 的可执行文件，并将其放入指定目录下。

2.接着生成并运行主程序 LProgram。

3.结果显示如下：

```
tos@tos211-vpc:~/Desktop$ ./Pro
Mainprogram Start:
生产者——时间 13:45:39 添加数据:87 缓存空间:87 0 0
生产者——时间 13:45:39 添加数据:87 缓存空间:87 87 0
消费者——时间 13:45:39 取出数据:87 缓存空间:0 87 0
消费者——时间 13:45:39 取出数据:87 缓存空间:0 0 0
生产者——时间 13:45:41 添加数据:16 缓存空间:0 0 16
消费者——时间 13:45:41 取出数据:16 缓存空间:0 0 0
生产者——时间 13:45:41 添加数据:16 缓存空间:16 0 0
消费者——时间 13:45:41 取出数据:16 缓存空间:0 0 0
生产者——时间 13:45:44 添加数据:36 缓存空间:0 36 0
消费者——时间 13:45:44 取出数据:36 缓存空间:0 0 0
生产者——时间 13:45:44 添加数据:36 缓存空间:0 0 36
消费者——时间 13:45:44 取出数据:36 缓存空间:0 0 0
生产者——时间 13:45:45 添加数据:93 缓存空间:93 0 0
消费者——时间 13:45:45 取出数据:93 缓存空间:0 0 0
生产者——时间 13:45:45 添加数据:93 缓存空间:0 93 0
消费者——时间 13:45:45 取出数据:93 缓存空间:0 0 0
生产者——时间 13:45:49 添加数据:22 缓存空间:0 0 22
消费者——时间 13:45:49 取出数据:22 缓存空间:0 0 0
生产者——时间 13:45:49 添加数据:22 缓存空间:22 0 0
消费者——时间 13:45:49 取出数据:22 缓存空间:0 0 0
生产者——时间 13:45:51 添加数据:28 缓存空间:0 28 0
生产者——时间 13:45:51 添加数据:28 缓存空间:0 28 28
消费者——时间 13:45:51 取出数据:28 缓存空间:0 0 28
消费者——时间 13:45:51 取出数据:28 缓存空间:0 0 0
Mainprogram Done!
```

6.心得体会

通过本次实验，基本掌握了在 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。

体会 4：看懂代码容易，但是想要使用相关函数编写程序还需要费点功夫。