实验2 生产者-消费者问题

班级： 07812201 学号： 1820221053 姓名： 曾泇睷

一、实验目的

独立设计并编写程序实现生产者-消费者问题，加深对进程通信的理解。

二、实验内容

1. 在Windows系统下实现生产者-消费者问题。

2. 具体要求如下：

(1) 创建一个包含6个缓冲区的缓冲池，初始为空，每个缓冲区能存放一个长度为10个字符的字符串。

(2) 创建2个生产者进程，要求每个生产者进程：

* + 随机等待一段时间后，向缓冲区中添加数据；
  + 若缓冲区已满，则等待消费者取走数据后再继续添加数据；
  + 重复添加数据12次。

(3) 创建3个消费者进程，要求每个消费者进程：

* + 随机等待一段时间后，从缓冲区中读取数据；
  + 若缓冲区为空，则等待生产者添加数据后再继续读取数据；
  + 重复读取数据8次。

(4) 输出显示每次添加或读取数据的时间，以及缓冲区的映像。

三、实验步骤

根据实验要求我们知道至少要完成以下概念的代码实现：缓冲池、生产者、消费者；同时我们还要规定相关的信号量。先给出相关宏变量定义（定义在头文件内）：

#define BUFFER\_LENGTH 11 //缓冲区长度

#define BUFFER\_COUNT 6 //6个缓冲区

#define PRODUCER\_COUNT 2 //2个生产者

#define PRODUCER\_REPEAT 12 //生产者重复12次

#define CONSUMER\_COUNT 3 //3个消费者

#define CONSUMER\_REPEAT 8 //消费者重复8次

在此处使用了Windows原生的API实现信号量P、V操作，其为：

WaitForSingleObject(hSemaphoreFull, INFINITE); //P 操作

ReleaseSemaphore(hSemaphoreEmpty, 1, nullptr); //V 操作

1. 构造缓冲池

// 缓冲区的结构

struct BufferPool {

char str[BUFFER\_COUNT][BUFFER\_LENGTH];

int head;

int tail;

};

1. 生产者产生随机字符串添加到缓冲区，若缓冲区已满则等待消费者取走缓冲区数据后继续添加数据。（此操作重复12次）

for (int i = 0; i < PRODUCER\_REPEAT; i++)

{

SleepTime = rand() % 1000;

Sleep(SleepTime); //休眠

WaitForSingleObject(hSemaphoreEmpty, INFINITE); // P(空信号量)：等待空信号量

WaitForSingleObject(hMutex, INFINITE); // P(互斥量)：等待互斥量

char\* s = pBuf->str[pBuf->tail]; // 获取缓冲区尾部的字符串指针

strcpy\_s(s, BUFFER\_LENGTH, randomString(10)); // 将生成的字符串复制到缓冲区

pBuf->tail = (pBuf->tail + 1) % BUFFER\_COUNT; // 更新尾部索引

…

}

1. 消费者从缓冲池取数据，若缓冲池为空则等待生产者添加数据后再取走数据。（此操作重复8次）

for (int i = 0; i < CONSUMER\_REPEAT; i++)

{

SleepTime = rand() % 2000;

Sleep(SleepTime); // 休眠

WaitForSingleObject(hSemaphoreFull, INFINITE); // P(满信号量)：等待满信号量

WaitForSingleObject(hMutex, INFINITE); // P(互斥量)：等待互斥量

// 从共享内存中消费一个字符串

char\* s = pBuf->str[pBuf->head]; // 获取缓冲区头部的字符串指针

memset(s, 0, sizeof(pBuf->str[pBuf->head])); // 清空消费的字符串

pBuf->head = (pBuf->head + 1) % BUFFER\_COUNT; // 更新头部索引

…

}

1. 在主进程创建2个生产者进程以及3个消费者进程，并创建信号

// 创建互斥量

HANDLE mutexHandle = CreateMutex(nullptr, FALSE, mutexName);

// 创建空信号量

HANDLE emptySemaphoreHandle = CreateSemaphore(nullptr, 6, 6, emptySemaphoreName);

// 创建满信号量

HANDLE fullSemaphoreHandle = CreateSemaphore(nullptr, 0, 6, fullSemaphoreName);

// 创建生产者进程

for (int i = 0; i < PRODUCER\_COUNT; i++) {

operationResult = CreateProcess(nullptr, producerExecutableName,nullptr, nullptr, TRUE, NORMAL\_PRIORITY\_CLASS, nullptr, nullptr, &startupInfoArray[i], &processInfoArray[i]);

}

// 创建消费者进程

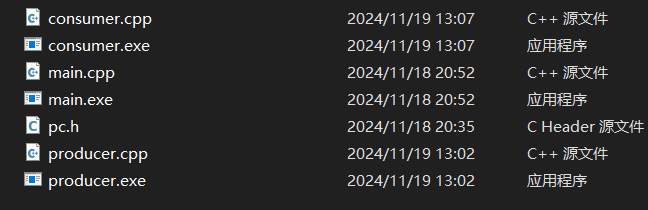
for (int i = PRODUCER\_COUNT; i < PRODUCER\_COUNT + CONSUMER\_COUNT; i++) {

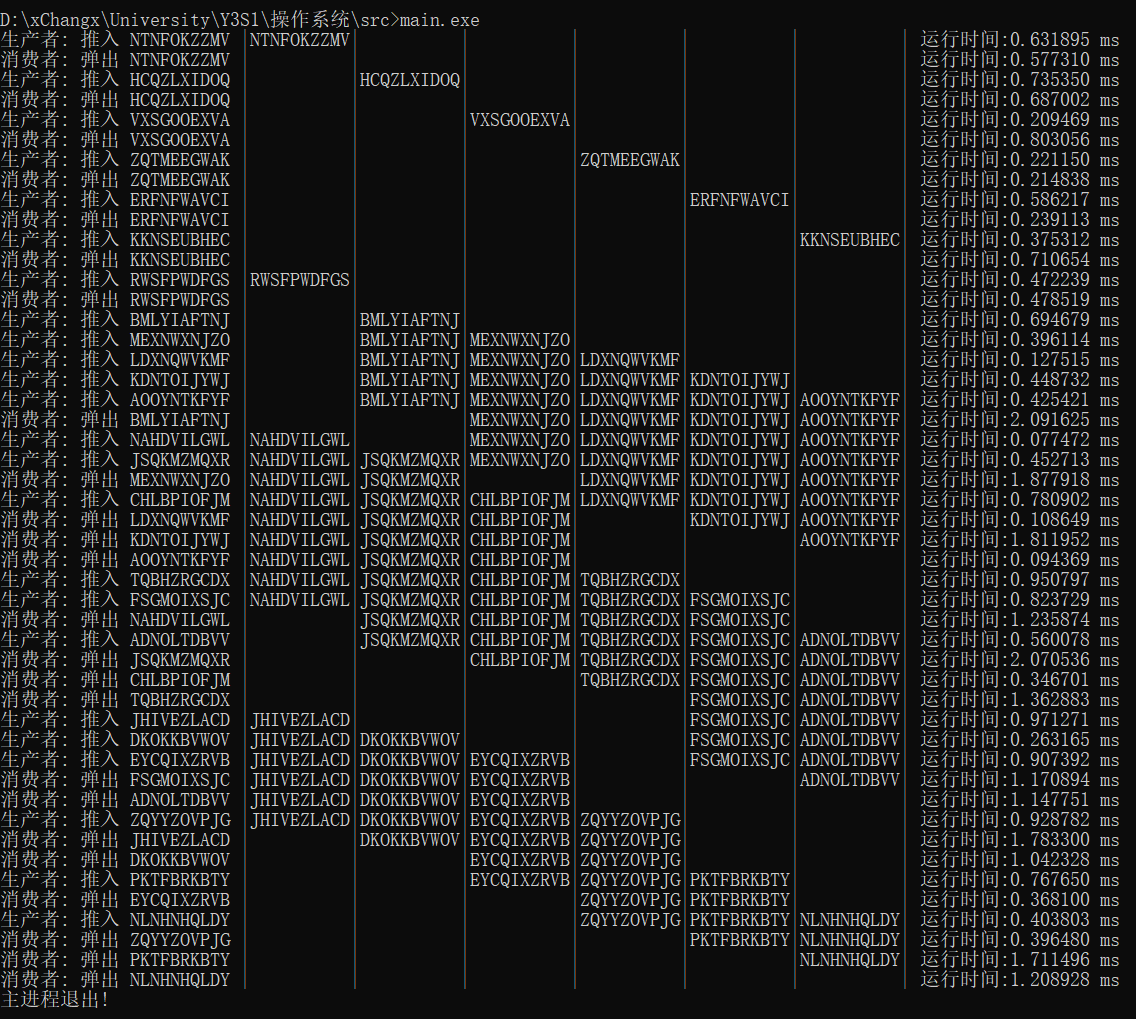
operationResult = CreateProcess(nullptr, consumerExecutableName, nullptr, nullptr, TRUE, NORMAL\_PRIORITY\_CLASS, nullptr, nullptr, &startupInfoArray[i], &processInfoArray[i]);

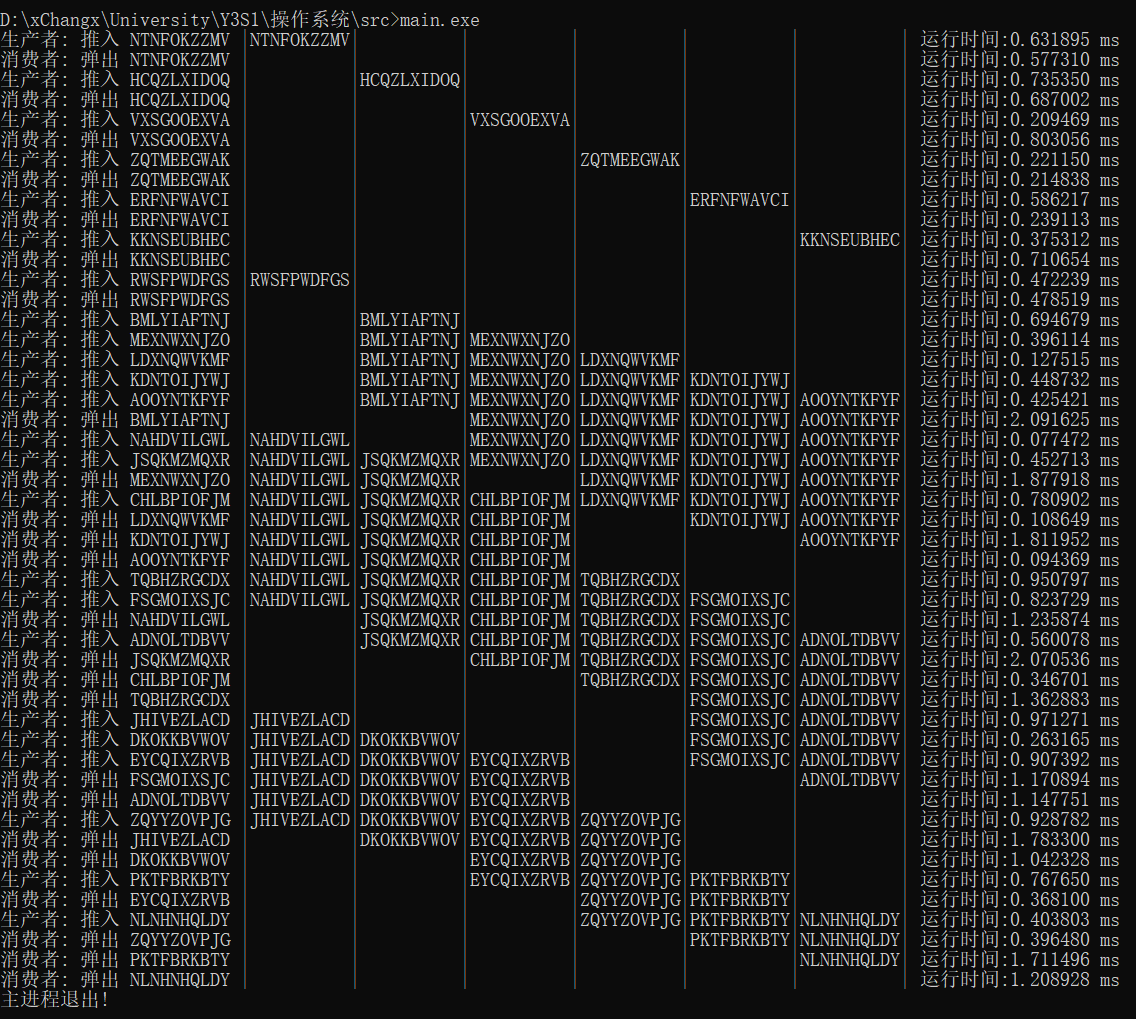
}

四、实验结果及分析

使用g++编译器分别编译源文件(.cpp)产生可执行文件(.exe)



在命令框执行main.exe，得到结果：



本次实验结果符合预期，并未出现异常现象，没有出现缓冲区读脏数据、重复写、死锁等现象。

五、实验收获与体会

通过本次实验，我深刻理解了生产者-消费者问题在操作系统进程通信中的应用。在 Windows 环境下，通过共享内存和信号量实现了进程间的同步与互斥，避免了资源竞争和数据不一致的问题。

此外，我也体会到了信号量机制的实际应用，使用 WaitForSingleObject 和 ReleaseSemaphore 实现了典型的 P/V 操作，掌握了缓冲区的同步与互斥控制方法；多进程间的通信，通过 CreateProcess 创建生产者和消费者进程，理解了如何使用 Windows API 实现进程间通信。

本次实验的完成，不仅巩固了操作系统理论知识，还提升了 C++ 编程技能，尤其是对 Windows API 的使用有了更直观的认识。

附录：程序清单及说明（黑体四号字）

宋体小四号字，行距22磅

（列出文件名及说明即可，不需要在此处复制代码，代码直接以源文件形式提供，但源文件中对代码要有必要的注释和说明）