燕山大学课程设计说明书

课程设计名称：操作系统

题目：基于 C/S 下的**多道程序缓冲区协调操作**

班级：计算机科学一班

开发小组名称：未命名
课题负责人：宋其翰
课题组成员：
宋其翰　100104010001　计算机科学一班　A

课题开发日期：2013 年 1 月

目录

1．概述

生产者-消费者问题是多线程程序设计的经典问题，也是对现实生活中相互合作的进程关系的一种抽象。在现在生活中，有很多情况都属于生产者-消费者问题，如 QQ 邮箱中的漂流瓶，打印作业等等。

本次课程设计在原有生产者和消费者基础上，增加了 2 个 buffer，如图 1 所示。Put 将数据放到 buffer1 中，move1 将 buffer1 中的数据移动到 buffer2 中，move2 将 buffer2 中的数据移动到 buffer3 中，最后在 buffer3 中再将数据 Get 出去。这样一来，在原有的生产者-消费者问题的基础上，又增加了 move1 和 move2 两个操作，进程间的同步操作显得尤为重要了。

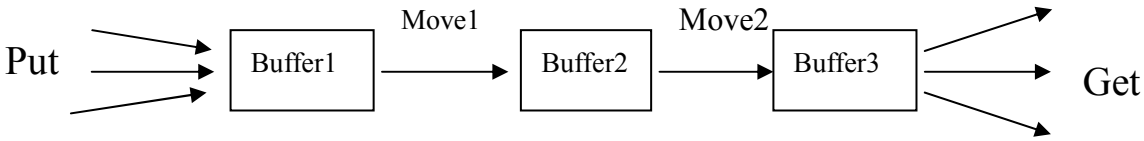

图 1 带有 move1 和 move2 的生产者-消费者问题

为了将本次课程设计更加贴近实用性，我将该题目中 Put 和 Get 数据换成了客户端，采用 C/S 模型的形式，基于 Win Socket 通的讯方式，传输层协议采用 TCP 协议，进一步提高了本程序的实用价值，基本上模拟了类似 QQ 邮箱漂流瓶的操作。程序设计如图 2 所示。

图 2 加入网络的生产者消费者问题

本次设计实用 VC6.0，使用 C 语言直接调用 windows API 的方式，分别编写出生产者端，消费者端和服务器端。生产者端主要负责 Put 数据至服务器，消费者端主要是从服务器 Get 数据，服务器端就实现处理和展示几个 buffer 的协同工作。为了演示方便，特在消费者端和服务器端增加了自动功能，可以自动生产和消费，这样演示起来就更加方便快捷，也可以在本地自己输入数据进行 put 和手动 get，以更加贴近实际用户操作。

2.使用的基本概念和原理

2.1 Windows Sockets

Windows Sockets 是 Windows 下网络编程的规范，它允许两个或者多个应用程序在相同机器上或者多台机器上，通过网络进行交流，是真正意义上的协议无关接口。其前身是以 U.C.Berkeley 大学[①]BSD UNIX 中流行的 Socket 接口为规范定义了一套 Microsoft Windows 下的网络编程接口。不但包含了 Berkeley Socket 风格的库函数，也有一组针对 Windows 的扩展函数，是程序员能够充分的利用 Windows 的消息驱动机制进行编程。Windows Sockets 规范定义了并记录了如何使用 API 和 Internet 协议（通常指 TCP/IP）协议，并且所有的 Windows Sockets 的实现都支持套接字和数据报套接字。

在 Windows Sockets 以两种模式执行 I/O 操作：阻塞和非阻塞，在阻塞模式下，执行 I/O 的 Windows Sockets 调用（如 send 和 recv）一直到操作完成才返回，而在非阻塞模式下，Windows Sockets 函数会立刻返回。阻塞套接字好处是简单，但是最常用的还是非阻塞模式的套接字。

2.2 线程以及 API 建立线程

线程，有时被称为轻量级进程(Lightweight Process，LWP)，是程序执行流的最小单元。另外，线程是进程中的一个实体，是被系统独立调度和分派的基本单位，线程自己不拥有系统资源，只拥有一点在运行中必不可少的资源，但它可与同属一个进程的其它线程共享进程所拥有的全部资源。一个线程可以创建和撤消另一个线程，同一进程中的多个线程之间可以并发执行。由于线程之间的相互制约，致使线程在运行中呈现出间断性。线程也有就绪、阻塞和运行三种基本状态。每一个程序都至少有一个线程，若程序只有一个线程，那就是程序本身。线程执行的工作大多数是运算工作，其不拥有资源，也是花费开销最小的实体。

在 Windows 上建立线程，使用的是 CreateThread 函数来创建一个线程进行，MSDN 上其函数原型如下所示：

HANDLE CreateThread(
LPSECURITY_ATTRIBUTES *lpThreadAttributes,* //安全性设置
DWORD *dwStackSize,* //设置初始栈的大小
LPTHREAD_START_ROUTINE *lpStartAddress,* //指向线程函数的指针
LPVOID *lpParameter,* //向线程函数传递的参数
DWORD *dwCreationFlags,* //线程标志立即执行还是等待
LPDWORD *lpThreadId);* //保存新线程的 id
返回值：

[①] U.C.Berkeley 大学即加利福尼亚大学伯克利分校。

创建线程成功后返回该线程的句柄，不成功就返回 FALSE。其第三个参数为指向函数的指针，其函数原型为：DWORD WINAPI ThreadProc (LPVOID lpParam);

但是在实际应用中，在某些情况下使用 CreateThread 建立进程、CloseHandle 关闭进程会造成线程的内存泄露，故微软不推荐使用这个函数来创建线程。微软推荐使用 _beginthreadex 来创建线程，_endthread 来销毁线程。其定义并不在 windows.h 中，故还需要在程序头包含 process.h 的头文件。_beginthread 的函数参数和 CreateThread 的参数相同，只不过其函数指针的原型为：static unsigned __stdcall ThreadProc(void * pM); 在这里就不详细叙述。

2.3 临界区以及临界区的建立

在支持多线程的操作系统中，两个或多个线程共享某些数据的情况并不罕见。例如，一个线程可以更新一个或者多个变量，而另一个线程可以使用这些变量。有时这会引发一个问题，有时又不会。

然而，假设线程共享几个变量或者数据结构。通常，这么多个变量或者结构的字段在它们之间必须是一致的。操作系统可以在更新这些变量的程序中间中断一个线程，那么使用这些变量的线程得到的将是不一致的数据。

结果是冲突发生了，并且通常不难想象这样的错误将对程序造成怎样的破坏。我们所需要的是类似于红绿灯的程序写作技术，以帮助我们对线程交通进行协调和同步，这就是临界区域。一个临界区域就是一块不可中断的程序代码。

有四个函数用于临界区域。要使用这些函数，必须定义一个临界区域对象，这是一个为 CRITICAL_SECTION 的全局变量。例如：

CRITICAL_SECTION cs ;

这个 CRITICAL_SECTION 数据型态是一个结构，但是其中的字段只能由 Windows 内部使用。这个临界区域对象必须先被程序中的某个线程初始化，通过：

InitializeCriticalSection (&cs) ;

来进行临界区的初始化。

当临界区域对象被初始化之后，线程可以通过下面的呼叫进入临界区域：

EnterCriticalSection (&cs) ;

在这时，线程被认为“拥有”临界区域对象。两个线程不可以同时拥有同一个临界区域对象，因此，如果一个线程进入了临界区域，那么下一个使用同一临界区域对象使用 EnterCriticalSection 的线程将在函数呼叫中被暂停。只有当第一个线程通过的离开临界区域时，函数才会传回控制权，离开临界区使用：

LeaveCriticalSection (&cs) ;

当临界区域不再被程序所需要时，可以通过可以删除临界区，删除临界区使用：

DeleteCriticalSection (&cs);

2.4 信号量以及信号量的建立

信号量(Semaphore)，有时被称为信号灯，是在多线程环境下使用的一种设施，是可以用来保证两个或多个关键代码段不被并发调用。在进入一个关键代码段之前，线程必须获取一个信号量；一旦该关键代码段完成了，那么该线程必须释放信号量。其它想进入该关键代码段的线程必须等待直到第一个线程释放信号量。

Windows API 中，建立信号量函数如下所示：

```
HANDLE CreateSemaphore(
  LPSECURITY_ATTRIBUTES lpSemaphoreAttributes,     //安全控制属性，一般为空
  LONG lInitialCount,                              //信号量的初始值
  LONG lMaximumCount,                              //信号量最大值
  LPCTSTR lpName );                                //信号量名称
```

若当前资源数量大于 0，表示信号量处于触发，等于 0 表示资源已经耗尽故信号量处于未触发。在对信号量调用等待函数时，等待函数会检查信号量的当前资源计数，如果大于 0(即信号量处于触发状态)，减 1 后返回让调用线程继续执行。一个线程可以多次调用 WaitForSingleObject 或者 WaitForMultipleObjects 函数来减小信号量。

当需要递增信号量时，使用的函数如下：

```
BOOL ReleaseSemaphore(
  HANDLE hSemaphore,                   //信号量句柄
  LONG lReleaseCount,                  //递增信号量个数
  LPLONG lpPreviousCount               //递增前信号量的个数
);
```

3 总体设计

3.1 总体设计思路

整体设计使用 VC++6.0 进行软件开发，使用 C 语言，结合 Windows API，进行面向过程的开发。并将一些常用的函数写到共同文件中，供 3 个程序共同使用。

3.2 总体结构

3.2.1 通讯消息结构定义

为了满足生产者端，消费者端，服务器端的通讯，需要定义一些通讯消息结构如下表一所示：

用途	结构

消息头部	**struct MSG_HEAD**{
	WORD *dwId;* //数据包类型
	DWORD *dwLength*; //数据包长度
	};

登陆信息包	**struct MSG_LOGIN**{
	char *szUsername[20];* //用户名
	WORD *wUserType;* //用户类型
	int *iCount;* //szUsername 长度
	};

登陆返回包	**struct MSG_LOGIN_RESP**{
	WORD *wbReturn;* //判断是否允许登陆
	};

递交和发送数据包	**struct MSG_PUT{** **DWORD** *dwLength;* //消息长度 **char** *szContent[256];* //消息 };

Get 消息包	**struct MSG_GET**{
	BOOL *bGet;*
	};

退出消息包	**struct MSG_EXIT**{
	WORD *wUserType;* //用户类型
	WORD *wExit;* //退出代码
	};

表一 定义的通讯包类型

定义了上述通讯包之后，定义了下述的一个结构体，在其中定义了一个 union 的结构体，将其作为消息类型的主干，而 MsgHead 中的 dwId 作为判断消息类型的依据。

```
struct myMSG{
    MSG_HEAD MsgHead;                    //消息头部
    union{
        MSG_LOGIN Login;                 //登陆
        MSG_LOGIN_RESP LoginResp;        //登陆返回包
        MSG_PUT PutMessage;              //消息包
        MSG_GET GetMessage;              //Get 消息
        MSG_EXIT msgExitCode;            //退出代码包
    };
};
```

同时也定义了一下常量作为 dwId 的判断依据：

#define *CMD_LOGIN* 0x81 //登陆包
#define *CMD_LOGIN_RESP* 0x82 //登陆返回包
#define *CMD_GET* 0x83 //消费者消费请求包
#define *CMD_PUT* 0x84 //数据包
#define *CMD_EXIT* 0x85 //退出包

3.2.2 接收函数定义

由于 WinSockets 中定义的 recv 函数只能接受定长的数据包，所以还需要定义一个 RecvPacket 的函数，用于接受不定长的数据包。并返回给调用函数进行判断，其定义为：

BOOL RecvPacket(
SOCKET *hSocket,* //socket
char **lpBuffer,* //用于接受的 buffer
UINT *dwSize);* //buffer 长度

其函数流程如下图所示：

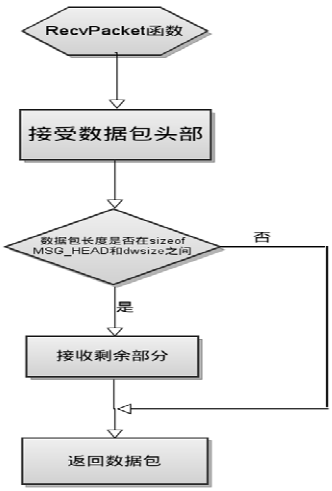

图 3 RecvPacket 函数流程

3.2.3 登陆函数定义

生产者登陆服务器端，使用的都是相同的函数，故将其放到一个函数内。定义为

BOOL Login_Server(
HWND *hMain,* //主窗口句柄
HWND *hListBox*, //listBox 窗口句柄
SOCKET *s,* //服务器端 socket
char **username,* //用户名
int *iUtype)*; //用户类型

该函数首先初始化句柄，然后使用阻塞模式发送登陆数据包，并接受返回的数据包。根

据返回的数据包，判断用户是否能够登陆。本程序为了演示，将所有的用户均可接受登陆。

3.3 服务器端创立线程

在服务器端，为了处理 WinSocket 的消息类型，要建立一个 ListenThread，用于监听客户端连接，本例中，所有客户端连接总数不能超过 10 个。并且每当连接上一个客户端，就需要建立一个服务线程 ServiceThread 来对该客户端的连接进行处理。

ServiceThread 处理连接如下图 4 所示：

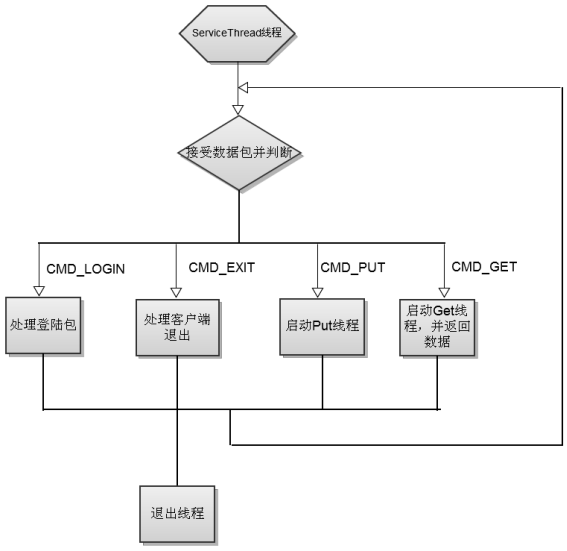

图 4 ServiceThread 线程处理流程图

3.4 服务器端线程同步

服务器端在 buffer 处理上有以下 4 个进程，其均为原子进程，不能被打断，Put(),Get()，move1(),move2()，为了保证这 4 个线程的同步，我使用了 6 个信号量，初始值和最大值如下表 2 所示：

信号量	初始值	最大值
hSIsPut	MAX_BUFF	MAX_BUFF
hSB1NULL	0	MAX_BUFF
hSB1FULL	MAX_BUFF	MAX_BUFF
hSB2NULL	0	MAX_BUFF
hSB2FULL	MAX_BUFF	MAX_BUFF
hSIsGet	0	MAX_BUFF

表 2 信号量的初值以及最大值

在服务器端，move1()和 move2()以 WM_TIMER 消息的形式响应。通过 SetTimer 设置时间，Put()和 Get()由相应的客户端提出响应。他们相互之前的同步关系如表 3 所示：

Put	Move1	Move2	Get
P(hSIsPut) Put(); V(hSB1NULL);	P(hSB1NULL) P(hSB1FULL) Move1(); V(hSIsPut); V(hSB2NULL);	P(hSB2FULL) P(hSB2NULL) Move2(); V(hSB1FULL); V(hSIsGet);	P(hSIsGet) Get(); V(hSB2FULL);

表 3 各进程之间的同步于互斥关系

4 编码设计

4.1 服务器端对 buffer 显示的处理

在服务器端定义了一个 buffer 的结构体，用于存储 buffer 的变量，定义如下：

```
typedef struct tagBuffer{
    HBRUSH bhr;          //定义画刷
    char content[256]; //buffer 内容
}Buffer;
```

在服务器端接收到 Put 之后，将其转化为 Buffer 变量，并使用 CreateSolidBrush(RGB(rand()%200+55,rand()%200+55,rand()%200+55))这样的一个函数创建一个非白色的画刷，将其放入 hbr 内供显示用。

当每次调用 move1()和 move2()移动完成之后，就使显示 buffer 状态的显示区域无效，重绘该区域。定义了一个 iflag[3][MAX_BUFF]变量，用于保存 buffer 状态的标志数组。当标志位为 1 时，调用随机生成的数组。当标志位为 0 时，调用白色画刷表示该显示区域为空。

4.2 生产者端自动生产

为了增加演示方便，在生产者端增加了一个自动生产的功能，从 resource.txt 文件中按行读取文件，并将其放入数据包中，每隔 1000ms 就发送一次。

读取 resource.txt 使用的是 ReadFile 函数，并将其保存到数组中随时使用。响应上也是使用 SetTimer 设置一个 1000ms 的定时器，响应 WM_TIMER 消息。

4.3 消费者端自动消费

消费者端自动消费也同生产者端一样，使用通过响应 WM_TIMER 消息，定时向服务器端发送 get 数据包即可取得消息。

5 测试时出现过的问题及其解决方法

5.1 包含工程外共同文件

为了保证代码复用，将一部分函数写到了一个共同的文件中并放到了工程文件夹的上层目录，将其加入了工程文件中，并在编译时使用#include "message.h"引用。在编译的时候都提示错误，找不到文件。后来经过思考发现文件路径不正确，后使用相对路径,将其替换成为#include "../message.h"，同时将#include "stdAfx.h"包含到每个文件头解决问题。

5.2 调试时线程同步不正确

为了跟踪线程的运行，win32 应用程序没有控制台，所以在调试时不知道怎么调试正确，最后通过查阅参考资料，在程序运行时启动控制台，可以在调试的适合程序向控制台输出一些调试信息，使之调试更加简单和容易。也跟踪出了一些由于粗心大意写错的手误，保证了程序在逻辑上的正确执行。

6 软件使用说明

6.1 软件的功能及运行环境

本软件是基于 C/S 模型的一个生产者-消费者问题，分为服务器端，生产者端和消费者端，用户可以使用用户名在相应客户端上进行登录，并进行相关操作。而服务器端部署在服务器上，可以用于监视 buffer 情况。运行环境为 Windows 2000,NT,2003,XP,7 版本的系统，安装 vc 运行库即可。

6.2 软件操作

6.2.1 生产者端

生产者端如图 5 所示，当服务器端运行后，可以输入用户名进行登录，登录完成后，可以点击菜单栏，选项->自动生产，软件就会自动从 resource.txt 读入内容生产出去，也可以从字符串端输入要生产的内容，点击生产之后送出。

图 5 生产者客户端界面

6.2.2 消费者端

生产者端如图 6 所示，当服务器端运行后，可以输入用户名进行登录，登录完成后，可以点击菜单栏，选项->自动状态，软件就会向服务器请求消费。也可以点击消费按钮之后，软件消费，并将消费结果显示在 listbox 上。

图 6 消费者客户端界面

6.2.3 服务器端

服务器端界面如图 7 所示，分为两部分，左边界面显示了当前客户端状态以及客户端的操作，右边显示了当前 buffer 的状态以及操作。Buffer 中有数据会以其他颜色显示，没数据就是以白色显示。

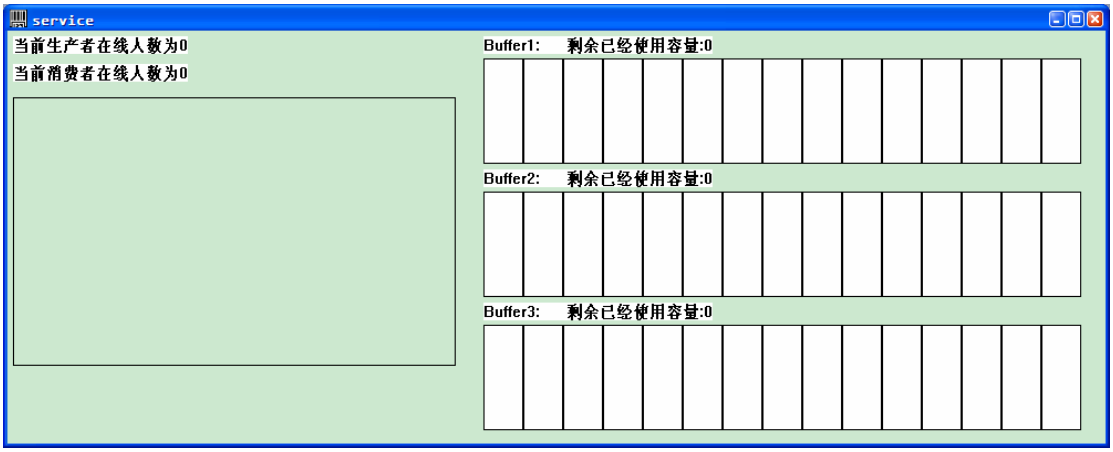

图 7 服务器端界面

7 总结

7.1 小组评定

宋其翰 A

7.2 收获和感谢

通过这次课程设计，我学会了进程之前的同步的做法，也通过学习，掌握了 Windows 编程的消息机制，Windows Sockets 编程的基本步骤和方法。掌握了调试 Windows API 程序的思路和方式。也学会了仔细检查，因为一个数字的写错，导致信号量差错，最后查了一天查出来了，还是自己细心程度不够。

这一路上走来，感谢穆运峰老师对我程序的大力支持，winsocket 函数就是学习的穆老师程序写出来的。感谢助教一周一来对我们的关心和辅导，也感谢申老师给我们上了一门精彩的操作系统课程。最后，也要谢谢父母对我长久一来的支持。

8 参考文献

[1] Charles Petzold.Windows 程序设计（第五版）[M].北京:北京大学出版社.2003
[2]罗云彬.Windows 环境下 32 位汇编语言程序设计(第二版)[M].北京：电子工业出版

社.2006

[3]代勇等.Visual C++网络通信编程技术详解[M].北京:机械工业出版社.2011

[4]王艳平.Windows 网络与通信程序设计[M].北京:人名邮电出版社.2006

[5]MoreWindows. 秒杀多线程第八篇 经典线程同步 信号量 Semaphore: http://blog.csdn.net/morewindows/article/details/7481609