《操作系统原理》实验报告

姓名	张旭	学号	U201817106	专业班级	软工 1805	时间	2020.04.23

一、实验目的

（1）理解操作系统线程的概念和应用编程过程；

（2）理解线程的同步概念和编程；

二、实验内容

（1）在 Ubuntu 或 Fedora 环境使用 fork 函数创建一对父子进程，分别输出各自的进程号和提示信息串。

（3）在 windows 环境下，利用高级语言编程环境（限定为 VS 环境或 VC 环境）调用 CreateThread 函数实现（2）的功能。

（4）在 windows 环境下，利用高级语言编程环境（限定为 VS 环境或 VC 环境）调用 CreateThread 函数和相关的同步函数，模拟实现“生产者-消费者”问题。

（5）在 windows 环境下，利用高级语言编程环境（限定为 VS 环境或 VC 环境）调用 CreateThread 函数实现“并发地画圆和画方”。圆的中心，半径，颜色，正方形的中心，边长，颜色等参数自己确定，合适就行。圆和正方形的边界上建议取 720 个点。为直观展示绘制的过程，每个点绘制后睡眠 0.2 秒~0.5 秒。

三、实验过程

（一）实验步骤

（1）进程信息

1. 使用 fork 创建子进程

2. 判断 pid，在父子进程中分别输出信息

```cpp
cppproject > test.cpp > main()
#include<iostream>
#include<unistd.h>
using namespace std;

int main()
{
    pid_t childPid;
    childPid = fork();
    if(childPid == 0)
    {
        cout << "我是子进程" << endl;
        cout << "子进程ID: " << getpid() << "父进程ID: " << getppid() << endl;
    }
    else
    {
        cout << "我是父进程" << endl;
        cout << "父进程ID: " << getpid() << endl;
    }

    return 0;
}
```

（3）Windows 平台实现多线程数数

1. 编写两个多线程 print 函数

```cpp
using namespace std;
DWORD WINAPI print1(LPVOID p)
{
    // LPVOID 是一个 void *,
    // 转为相应类型的指针  int * a = (int
    char* name = (char*)p;
    for (int i = 1; i <= 1000; i++)
    {
        cout << name << ":" << i << endl;
        Sleep(500);
    }
    return 0;
}
```

```cpp
DWORD WINAPI print2(LPVOID p)
{
    // LPVOID 是一个 void *,
    // 转为相应类型的指针  int * a = (int *) n;
    char* name = (char*)p;
    for (int i = 1000; i >= 1; i--)
    {
        cout << name << ":" << i << endl;
        Sleep(500);
    }
    return 0;
}
```

2. 编写主函数，使用 createThread 创建两个线程

```cpp
int main()
{
    HANDLE th[2];

    th[0] = CreateThread(NULL, 0, print1, (LPVOID)"thread1", 0, NULL);
    th[1] = CreateThread(NULL, 0, print2, (LPVOID)"thread2", 0, NULL);
    WaitForMultipleObjects(2, th, true, INFINITE);
    CloseHandle(th[0]);
    CloseHandle(th[1]);

    return 0;
```

3. 运行代码，得到结果

（4）生产者-消费者问题

1. 首先定义了临界区（互斥量）、full 和 empty 信号量。同时使用 vector 定义了容器。

```
const int MAXN = 5;
const int THREAD_NUM = 4;
vector<int> container;  // 容器
CRITICAL_SECTION cs;    // 临界区
HANDLE Empty, Full;     // 信号量
```

2.根据课堂上讲的编写生产者和消费者函数

```
DWORD WINAPI produce(LPVOID p)
{
    int id = *(int*)p;
    while (true)
    {
        WaitForSingleObject(Empty, INFINITE
        EnterCriticalSection(&cs);
        container.push_back(10);
        Sleep(500);
        cout << "编号：" << id << ", 生产了
        LeaveCriticalSection(&cs);
        ReleaseSemaphore(Full, 1, NULL);
    }
}
```

```
DWORD WINAPI consume(LPVOID p)
{
    int id = *(int*)p;
    while (true)
    {
        WaitForSingleObject(Full, INFINITE);
        EnterCriticalSection(&cs);
        Sleep(500);
        cout << "编号：" << id << ", 消费了第" <<
        container.pop_back();
        LeaveCriticalSection(&cs);
        ReleaseSemaphore(Empty, 1, NULL);
    }
}
```

记得在容器中添加删除

3.编写主函数

创建信号量，初始化临界区。

创建两个生产者两个消费者，使用位运算判断。

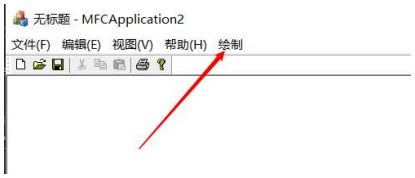
```
{
    InitializeCriticalSection(&cs);
    Empty = CreateSemaphore(NULL, MAXN, MAXN, NULL);
    Full = CreateSemaphore(NULL, 0, MAXN, NULL);
    HANDLE thread[THREAD_NUM];
    int ID[THREAD_NUM];
    for (int i = 0; i < THREAD_NUM; i++)
    {
        ID[i] = i + 1;
        if (i & 1)
            thread[i] = CreateThread(NULL, 0, produce, &ID[i], 0, NULL);
        else
            thread[i] = CreateThread(NULL, 0, consume, &ID[i], 0, NULL);
    }

    WaitForMultipleObjects(THREAD_NUM, thread, true, INFINITE);
    for (int i = 0; i < THREAD_NUM; i++)
    {
        CloseHandle(thread[i]);
    }

    CloseHandle(Empty);
    CloseHandle(Full);
    DeleteCriticalSection(&cs);
    return 0;
}
```

4.运行代码，实验结果

（5）并发画圆画方

1.创建 mfc 项目，生成代码

2.添加菜单项目



3.编写响应函数

画圆

```
UINT DrawCircle(LPVOID lp)
{
    CMFCApplication2View* pDlg = (CMFCApplication2View*)lp;
    CDC* pDC = pDlg->GetDC();

    CPoint CirclePoint(200, 200);
    COLORREF crColor = RGB(255, 0, 0);
    CPen pen(PS_SOLID, 5, crColor);
    pDC->SelectObject(&pen);
    int r = 150;
    const double degree_half = 0.008726; //每半度 2×3.1415 / 720

    for (int i = 0; i < 720; i++)
    {
        pDC->SetPixelV(
            CirclePoint.x + r * sin(degree_half * i),
            CirclePoint.y + r * cos(degree_half * i),
            crColor);
        Sleep(10);
    }
    return 0;
}
```

画方

```
// CMFCApplication2View 消息处理程序
UINT DrawRectgle(LPVOID lp)
{
    CMFCApplication2View* pDlg = (CMFCApplication2View*)lp;
    CDC* pDC = pDlg->GetDC();

    CPoint LeftRight(400, 50);
    COLORREF crColor = RGB(255, 0, 0);
    CPen pen(PS_SOLID, 5, crColor);
    pDC->SelectObject(&pen);
    const int width = 2;
    for (int i = 0; i < 180; i++)
    {
        pDC->SetPixelV(
            LeftRight.x + width * i,
            LeftRight.y,
            crColor);
        Sleep(10);
    }

    for (int i = 0; i < 180; i++)
    {
        pDC->SetPixelV(
            LeftRight.x + width * 180,
            LeftRight.y + width * i,
            crColor);
        Sleep(10);
    }

    for (int i = 0; i < 180; i++)
    {
        pDC->SetPixelV(
            LeftRight.x - width * i + width * 180,
            LeftRight.y + w
            crColor);
        Sleep(10);
    }
```

(局部变量) const int width = 2
联机搜索

```
for (int i = 0; i < 180; i++)
{
    pDC->SetPixelV(
        LeftRight.x,
        LeftRight.y + width * 180 - width * i,
        crColor);
    Sleep(10);
}

return 0;
```

响应创建线程

```
void CMFCApplication2View::Ondraw()
{
    // TODO: 在此添加命令处理程序代码
    this->UpdateWindow();
    pThread_Circle = AfxBeginThread(DrawCircle, this);
    pThread_Circle = AfxBeginThread(DrawRectgle, this);

}
```

（二）解决错误和优化

实验（1）和实验（3）比较简单没有出现什么问题，实验（4）（5）的问题如下：

1.【语法错误】实验（4）中创建线程出现问题。

错误现象：线程的 ID 全部都是 4.

解决方法：createThread 中传入线程的名字时出现问题。之前是&（i+1）.由于传的是地址，所以所有的线程名字的地址都是一样的，在循环中 i 自增到 4，所以线程的 ID 全部是 4。最后修改的时候，建了一个数组，每个线程使用不同元素的地址作为 id 地址。

2.【操作过程错误】实验（5）中使用了 MFC

由于限制在限定为 VS 环境或 VC 环境，所以选择并不多。有的人选择使用的是 easyx 图形库。我最终选择的是 MFC，由于对 MFC 编程并不熟悉，所以编码时遇到很多困难。

错误描述：添加菜单后，点击后无响应。

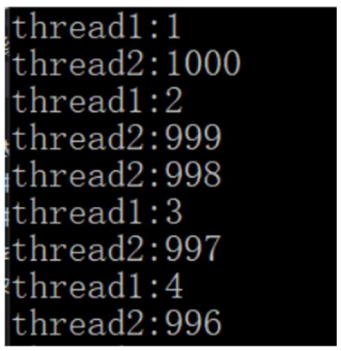
解决方法：检查逻辑，发现项目中逻辑有问题，修改代码后正常。

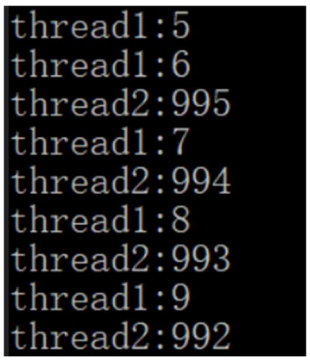
四、实验结果

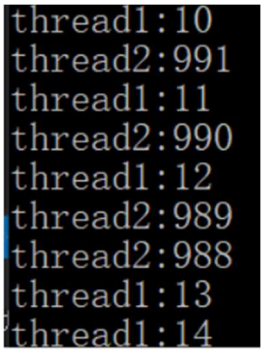
（1）进程信息

在子进程中也显示一遍父进程 ID，发现相同，正确

（3）Windows 平台实现多线程数数

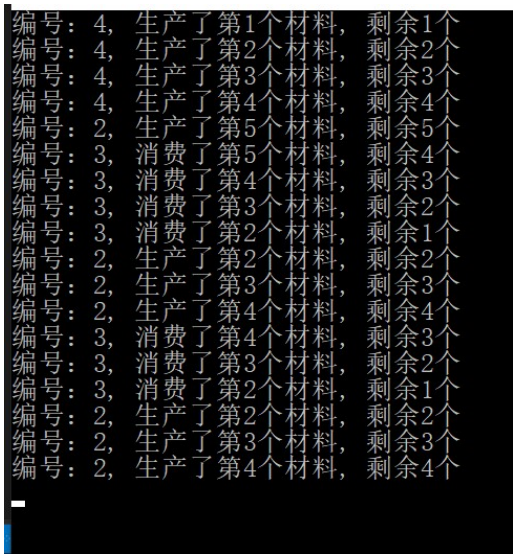






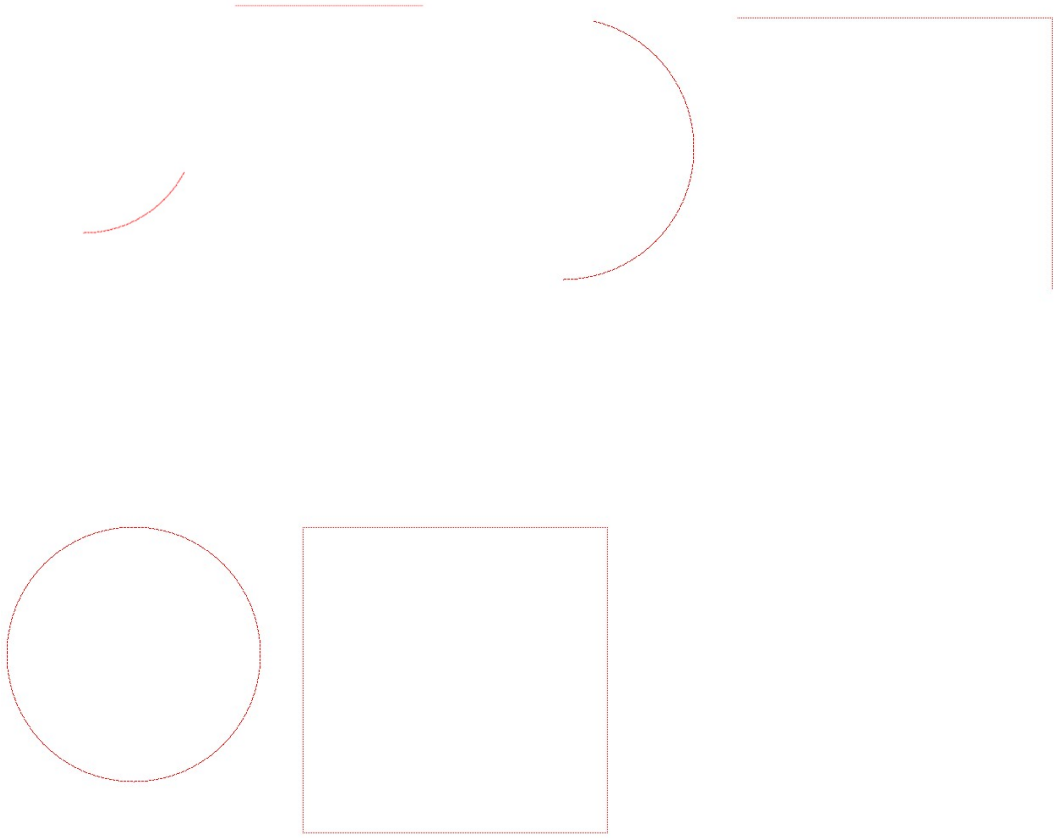
数数的大小顺序正确

（4）生产者-消费者问题



分析生产消费顺序，产品数量，发现结果正确

（5）并发画圆画方



观察画圆画方，的确并发运行

五、体会

通过本次实验，我对操作系统线程的概念和生成过程、操作系统线程的同步概念有了更深刻的理解，掌握了如何在 Linux 与 Windows 下创建线程，如何处理线程之间的关系，对操作系统的功能与原理有了进一步的了解。并且学习了基础的 MFC 编程，编程能力得到了提升。