《计算机操作系统》实验报告

实验题目：死锁观察与避免

姓名：汪雨卿　　学号：19120191　　实验日期：2021.12.07

实验环境：

Visual Studio 2019; C++

实验目的：

死锁会引起计算机工作僵死，造成整个系统瘫痪。因此，死锁现象是操作系统特别是大型系统中必须设法防止的。学生应独立的使用高级语言编写和调试一个系统动态分配资源的简单模拟程序，观察死锁产生的条件，并采用适当的算法，有效的防止死锁的发生。通过实习，更直观地了解死锁的起因，初步掌握防止死锁的简单方法，加深理解课堂上讲授过的知识。

实验内容：

（1） 设计一个 n 个并发进程共享 m 个系统资源的系统。进程可动态地申请资源和释放资源。系统按各进程的申请动态地分配资源。

（2） 系统应能显示各进程申请和释放资源以及系统动态分配资源的过程，便于用户观察和分析。

（3） 系统应能选择是否采用防止死锁算法或选用何种防止算法（如有多种算法）。在不采用防止算法时观察死锁现象的发生过程。在使用防止死锁算法时，了解在同样申请条件下，防止死锁的过程。

操作过程：

1. 设计算法的流程图

图 1 调度算法逻辑框图中展示了本程序的算法逻辑过程。

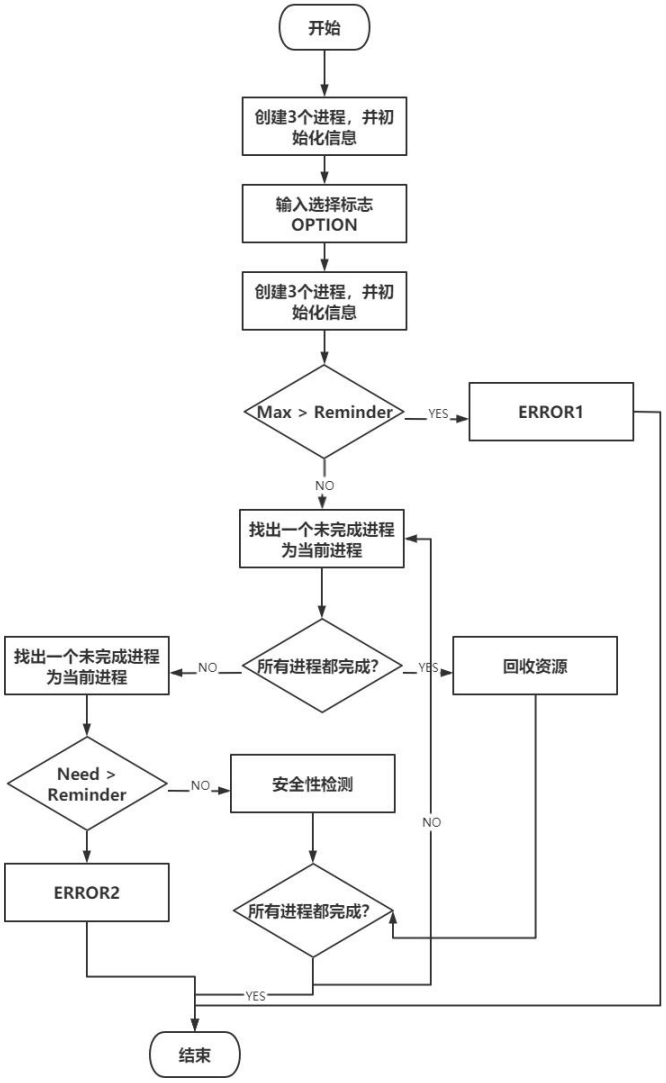

图 1 调度算法逻辑框图

2. 设计进程控制块 PCB 的数据结构 和 全局变量

1. 将进程控制块的 PCB 用类来表达，设置 PID，Finish，Max， Allocation,Need,Flag

- PID：标志进程块的序号
- Finish：标志进程是否完成（TRUE 完成，False 未完成）
- Max：每个进程需要分配的最多的设备数目
- Allocation：该进程已分配到的设备数目
- Need：该进程还需分配的设备数目
- Flag：在安全性算法中，flag 标志该进程是否被分配。

2. 全局变量

- Advance：标记所有结束
- reminder：记录 OS 剩余可分配的设备总数
- Option：记录是否采用死锁解决算法

图 2 PCB 类和全局变量 实现中展示了具体代码实现。

```
bool Advance = true;    // 所有进程结束, = false;进程未结束, = true
int reminder = 10;      // reminder=剩余设备的总个数
bool Option;            // 是否采用死锁解决算法

// 定义PCB进程块的数据结构，以及对应的getter和setter方法
class PCB               // 定义PCB进程的数据结构
{
private:
    int PID;            // 进程控制块ID
    bool Finish;        // 进程完成标志; True = 完成, False 未完成

public:
    vector<int> Claim;          // 动态申请的进程数 (1,2,3,4)
    int Max;                    // 每个进程的最大需要的设备个数 总和
    int Allocation;             // 以及分配到的设备数 (已经有的个数)
    int Need;
    bool Flag;                  // 安全性算法，记录是否finish

    PCB();
    void setPCB(int pid, int all_t, int state, bool finish);
    int getPID();
    bool getFinish();
    void setFinish(bool finish);
};
```

图 2 PCB 类和全局变量

3. 银行家算法的具体实现

注释：本代码中调入进程，采用了时间片轮转的方法。

a. 第一次判断：判断每个进程所需要的最大设备数是否比 OS 可分配的最大资源数小。当结果为否定时，输出报错，退出程序。

例如：一个操作系统最多可分配的设备资源为 10 个，而进程 a 所需要的最大资源数为 11。则会输出报错信息。

图 3 中为具体的代码实现。

```
cout << endl << "==各进程块需要的最大资源数 " << endl;
// 输出所有进程，最大需要的进程序列
for (int i = 1; i <= 3; i++)
{
    int max = pcb[i-1].Max;
    cout << "  PCB" << i << " Max:  " << left << setw(5) << max << endl;
    if (max > 10)
    {
        cout << "------------------------------------------------------" << endl;
        cout << "ERROR1: 进程" << i << "需要资源数大于资源现有最大资源数。" << endl;
        cout << "------------------------------------------------------" << endl;
        cout << endl;
        return;
    }
}
```

图 3 分配资源数和系统资源数比较代码

b.利用安全性算法以及死锁处理：判断每一次分配的结果是否是安全的。若不安全，则先判断是否使用死锁处理程序；若使用，则进入死锁处理流程，若不使用，则报错退出程序。

安全性算法判断不安全的可能有以下几点：

1. Require > 提出的 Need
2. Require > Work
3. 分配后，进程死锁

图 4-5 中为具体的代码实现。

```
// 时间片轮转调度进程
int count = pcb.size();             // 初始化count= 待运行进程的个数
cout << left << setw(10) << "PID" << setw(10) << "Allocation" << setw(10) << "Available" << endl;
while (count)
{
    it = pcb.begin();

    for (int m = 0; m < pcb.size();m++) {       // 初始化pcb中Need的大小
        pcb[m].Need = pcb[m].Max - pcb[m].Allocation;
    }

    if ((*it).Claim[0] > reminder)  // 1. 申请资源数 > 分配资源数，报错/进行死锁处理
    {
        cout << "------------------------------------------------------" << endl;
        cout << "ERROR2: 申请资源数 > 可分配资源数。" << endl;
        cout << "------------------------------------------------------" << endl;
        cout << endl;
        if (Option)
        {
            // 使用进程死锁处理，进入程序
            PCB temp = *it;
            pcb.erase(pcb.begin());
            pcb.push_back(temp);
            cout << (*pcb.begin()).Claim[0] << endl;
        }
        else {
            return; // 没有进程死锁处理，则直接退出
        }
    }
    else
```

图 4 安全性算法 1

```
    //分配资源
    reminder -= (*it).Claim[0];
    (*it).Allocation += (*it).Claim[0];
    (*it).Claim.erase((*it).Claim.begin());
    cout << left << setw(10) << (*it).getPID()  << setw(10) << (*it).Allocation << setw(10) << reminder << endl;
    if ((*it).Claim.size() == 0)
    {
        count -= 1;
        (*it).setFinish(true);      // 设置进程
        cout << "-------------进程" << (*it).getPID() << "运行完毕-----------" << endl;
        pcb.erase(pcb.begin());
        continue;
    }
    else
    {
        PCB temp = *it;
        pcb.erase(pcb.begin());
        pcb.push_back(temp);
    }
```

图 5 安全性算法 2

4. 初始条件设置

a. 测试 1

OPTION ： 1

时间片：4

PCB1: 1 1 1 2

PCB2: 2 2 1 3

PCB3: 5 6 7 3

a. 测试 2

OPTION ：1

时间片：3

PCB1: 3 2 1

PCB2: 1 2 3

PCB3：6 2 1

结果：

```
===============开启 银行家算法===============

===============选项：是否需要选用“防止死锁”算法===============
1. 选用“防止死锁”算法
0. 不用“防止死锁”算法
1
OPTION：选用“防止死锁”算法

=============== 进程块信息 ================

===各进程块需要的资源序列
  PCB1 :  1     1     1     2
  PCB2 :  2     2     1     3
  PCB3 :  5     6     7     3

==各进程块需要的最大资源数
  PCB1 Max:  5
  PCB2 Max:  8
  PCB3 Max:  21
------------------------------------------------------------
ERROR1：进程3需要资源数大于资源现有最大资源数。
------------------------------------------------------------
```

```
===============开启 银行家算法===============

===============选项：是否需要选用“防止死锁”算法===============
1. 选用“防止死锁”算法
0. 不用“防止死锁”算法
1
OPTION：选用“防止死锁”算法

=============== 进程块信息 ================

===各进程块需要的资源序列
  PCB1 :  3     2     1
  PCB2 :  1     2     3
  PCB3 :  6     2     1

==各进程块需要的最大资源数
  PCB1 Max:  6
  PCB2 Max:  6
  PCB3 Max:  9
PID       AllocationAvailable
1         3         7
------------------------------------------------------------
ERROR3：算法不安全
------------------------------------------------------------
```

体会：

本次实验帮助我更好的了解了 OS 进程死锁解决的银行家算法的实现过程，算法逻辑。同时也将最开始停留在理论上的银行家算法，通过代码的方式，在计算机在实现。在死锁处理方式上，上个学期我只知道有哪些方法，但是并没有思考过如何设计相关的代码，也没有思考过时间片轮转和银行家算法的结合。而这次的实验更好的帮组我融合贯通了 OS 进程调度，死锁处理的知识点，帮助我更好的学习了操作系统的知识。此外，在未来的学习中我也应该更多的思考，并且实践，巩固和贯通自己学习过的内容。