**广州大学学生实验报告**

**开课学院及实验室：**计算机科学与网络工程学院软件实验室 **2023年 5月 日**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **学院** | **计算机科学与网络工程学院** | **年级/专业/班** | **软件211** | **姓名** | 张景致 | **学号** | 32106300004 |
| **实验课程名称** | 操作系统实验 | | | | | **成绩** |  |
| **实验项目名称** | **银行家算法** | | | | | **指导老师** |  |

**实验二 银行家算法**

1. **实验目的**

1、了解什么是操作系统安全状态和不安全状态；

2、了解如何避免系统死锁；

3、理解银行家算法是一种最有代表性的避免死锁的算法，掌握其实现原理及实现过程。

1. **实验环境**

Linux archlinux 6.3.1-zen1-1-zen #1 ZEN SMP PREEMPT\_DYNAMIC Mon, 01 May 2023 17:42:12 +0000 x86\_64 GNU/Linux

clang version 15.0.7

Target: x86\_64-pc-linux-gnu

Thread model: posix

InstalledDir: /usr/bin

1. **实验内容**

根据银行家算法的基本思想，编写和调试一个实现动态资源分配的模拟程序，并能够有效避免死锁的发生。

1. **实验原理**

进程申请资源时，系统通过一定的算法判断本次申请是否不可能产生死锁（处于安全状态）。若可能产生死锁（处于不安全状态），则暂不进行本次资源分配，以避免死锁。算法有著名的银行家算法。

**1、什么是系统的安全状态和不安全状态？**

所谓安全状态，是指如果系统中存在某种进程序列＜P1，P2，…，Pn＞，系统按该序列为每个进程分配其所需要的资源，直至最大需求，则最终能使每个进程都可顺利完成，称该进程序列＜P1，P2，…，Pn，＞为安全序列。

如果不存在这样的安全序列，则称系统处于不安全状态。

**2、银行家算法**

把操作系统看作是银行家，操作系统管理的资源相当于银行家管理的资金，进程向操作系统请求分配资源相当于用户向银行家贷款。

**为保证资金的安全，银行家规定:**

(1) 当一个顾客对资金的最大需求量不超过银行家现有的资金时就可接纳该顾客;

(2) 顾客可以分期贷款，但贷款的总数不能超过最大需求量;

(3) 当银行家现有的资金不能满足顾客尚需的贷款数额时,对顾客的贷款可推迟支付,但总能使顾客在有限的时间里得到贷款;

(4) 当顾客得到所需的全部资金后,一定能在有限的时间里归还所有的资金。

**操作系统按照银行家制定的规则设计的银行家算法为：**

（1）进程首次申请资源的分配：如果系统现存资源可以满足该进程的最大需求量，则按当前的申请量分配资源，否则推迟分配。

（2）进程在执行中继续申请资源的分配：若该进程已占用的资源与本次申请的资源之和不超过对资源的最大需求量，且现存资源能满足该进程尚需的最大资源量，则按当前申请量分配资源，否则推迟分配。

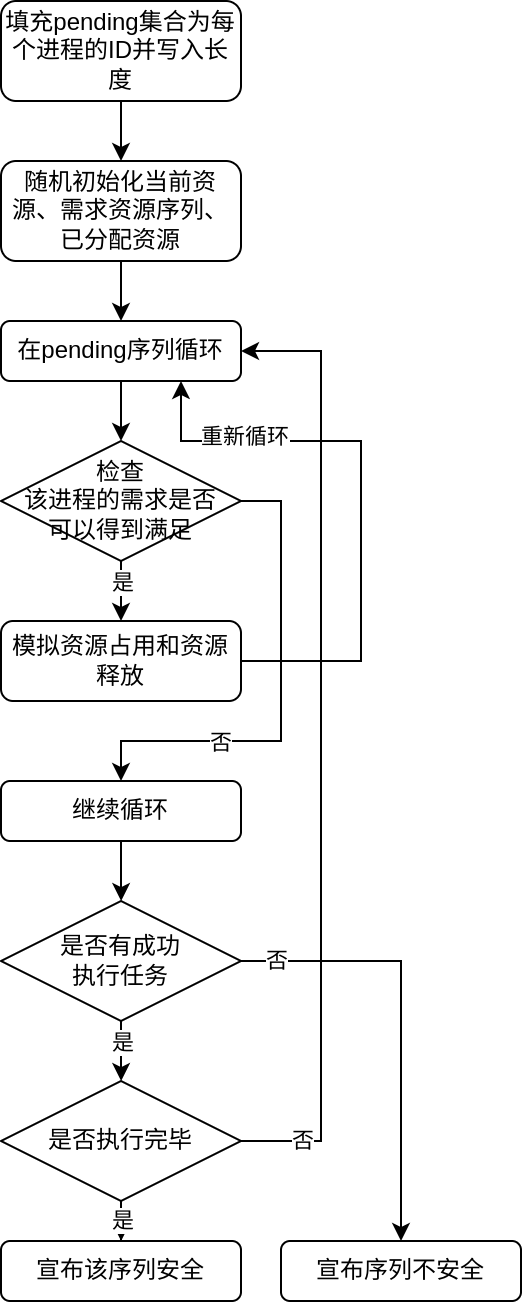
（3）至少一个进程能完成：在任何时刻保证至少有一个进程能得到所需的全部资源而执行到结束。

银行家算法通过动态地检测系统中资源分配情况和进程对资源的需求情况来决定如何分配资源，并能在确保系统处于安全状态时才把资源分配给申请者，从而避免系统发生死锁。

1. **实验中用到的系统调用函数**

因为是模拟程序，可以不使用系统调用函数。

1. **实验步骤**
2. 画出银行家算法流程图；



1. 对算法所用的数据结构进行说明；

该算法中每个进程为一个对象单位，包含：当前已占用资源allocated、共需求资源requirement。除此之外还有代表当前空闲资源的available和记录余下需要使用的进程的pending顺序表。

1. 测试数据随机产生。不可手工输入
2. 编写程序并调试
3. 多次测试程序，截屏输出实验结果
4. 根据实验结果与理论课讲述的原理进行实验分析
5. **实验结果分析（截屏的实验结果，与实验结果对应的实验分析）**

1、实验结果与实验程序、实验步骤、实验原理、操作系统原理的对应分析；

--- INITIALIZATION ---

FREE P0 P1 P2 P3 P4 P5 P6 P7 P8 P9

R0 6 0/1 12/18 0/0 8/13 17/19 15/16 10/11 1/12 3/18 9/16

R1 12 2/6 1/9 1/8 0/2 1/3 9/16 10/11 2/5 4/6 3/8

R2 7 7/18 1/2 1/2 0/1 9/14 5/14 0/3 0/0 0/2 0/6

R3 15 5/6 7/8 3/16 8/15 3/11 0/0 2/12 1/3 4/9 0/18

R4 18 13/17 2/14 8/14 6/15 0/6 2/6 3/7 3/6 0/0 0/16

R5 4 3/17 3/4 2/7 9/12 13/18 2/11 0/3 1/11 9/11 8/10

R6 8 1/17 2/5 13/15 4/10 9/15 3/15 5/12 9/14 4/5 2/17

R7 4 5/16 3/8 0/0 5/8 4/9 7/9 1/6 4/18 11/12 7/16

R8 17 0/2 0/1 0/8 4/8 11/13 1/3 11/12 2/7 12/13 6/7

R9 4 13/16 6/15 3/12 0/0 0/1 0/1 0/0 13/17 1/10 1/9

---EXECUTION OF P3---

R0 R1 R2 R3 R4 R5 R6 R7 R8 R9

BEFORE 6 12 7 15 18 4 8 4 17 4

AFTER 14 12 7 23 24 13 12 9 21 4

---EXECUTION OF P4---

R0 R1 R2 R3 R4 R5 R6 R7 R8 R9

BEFORE 14 12 7 23 24 13 12 9 21 4

AFTER 31 13 16 26 24 26 21 13 32 4

---EXECUTION OF P0---

R0 R1 R2 R3 R4 R5 R6 R7 R8 R9

BEFORE 31 13 16 26 24 26 21 13 32 4

AFTER 31 15 23 31 37 29 22 18 32 17

---EXECUTION OF P7---

R0 R1 R2 R3 R4 R5 R6 R7 R8 R9

BEFORE 31 15 23 31 37 29 22 18 32 17

AFTER 32 17 23 32 40 30 31 22 34 30

---EXECUTION OF P6---

R0 R1 R2 R3 R4 R5 R6 R7 R8 R9

BEFORE 32 17 23 32 40 30 31 22 34 30

AFTER 42 27 23 34 43 30 36 23 45 30

---EXECUTION OF P5---

R0 R1 R2 R3 R4 R5 R6 R7 R8 R9

BEFORE 42 27 23 34 43 30 36 23 45 30

AFTER 57 36 28 34 45 32 39 30 46 30

---EXECUTION OF P8---

R0 R1 R2 R3 R4 R5 R6 R7 R8 R9

BEFORE 57 36 28 34 45 32 39 30 46 30

AFTER 60 40 28 38 45 41 43 41 58 31

---EXECUTION OF P9---

R0 R1 R2 R3 R4 R5 R6 R7 R8 R9

BEFORE 60 40 28 38 45 41 43 41 58 31

AFTER 69 43 28 38 45 49 45 48 64 32

---EXECUTION OF P2---

R0 R1 R2 R3 R4 R5 R6 R7 R8 R9

BEFORE 69 43 28 38 45 49 45 48 64 32

AFTER 69 44 29 41 53 51 58 48 64 35

---EXECUTION OF P1---

R0 R1 R2 R3 R4 R5 R6 R7 R8 R9

BEFORE 69 44 29 41 53 51 58 48 64 35

AFTER 81 45 30 48 55 54 60 51 64 41

从图上来看，此执行绪是安全的，并且安全序列为3407658921，每一个任务得到分配执行的情况均已列出，未检测到死锁风险。

--- INITIALIZATION ---

FREE P0 P1 P2 P3 P4 P5 P6 P7 P8 P9

R0 19 0/12 16/17 3/14 1/13 16/17 9/11 0/18 0/0 1/3 3/12

R1 7 1/4 0/0 8/18 5/17 2/12 0/4 0/2 1/9 11/14 10/13

R2 15 7/16 4/11 11/15 3/4 6/7 1/6 1/9 1/12 11/17 0/5

R3 11 4/13 11/16 1/2 1/2 1/2 6/13 6/17 13/19 4/6 0/8

R4 14 10/13 8/14 2/16 2/3 2/3 7/19 10/13 4/12 0/2 1/2

R5 2 0/5 12/13 4/11 3/13 2/17 5/6 1/3 1/2 2/15 0/3

R6 0 0/3 0/0 2/3 3/7 3/18 4/14 0/1 4/18 1/3 9/18

R7 19 9/17 0/1 0/2 1/16 3/6 0/12 13/18 1/6 5/14 10/15

R8 16 17/19 1/5 13/14 1/15 5/8 4/5 0/1 1/2 1/13 3/7

R9 15 7/18 2/6 4/14 16/17 5/12 1/9 7/12 12/19 6/13 6/9

---EXECUTION OF P1---

R0 R1 R2 R3 R4 R5 R6 R7 R8 R9

BEFORE 19 7 15 11 14 2 0 19 16 15

AFTER 35 7 19 22 22 14 0 19 17 17

Current state is unsafe!

从图上所示，此次生成的序列之中由于R6资源开始尚无余量，只有对此无需求的P1被顺利分配，在此之后所有其他进程均无法被满足，因此造成了死锁风险。

1. 不同条件下的实验结果反应的问题及原因；

经过分析实践得，该实验每次执行是否有“安全序列”产生还是会发生死锁风险与几个随机参数关系较大。

#define **COUNT\_RESOURCE** 10  
#define **COUNT\_PROCESS** 10  
#define **INITIAL\_CAP** 20  
#define **DEMAND\_CAP** 20

顾名思义，这些超参数影响了生成的数值特征。明显的，更高的initial cap将会产生更多的安全序列，而更高的demand cap会给予某些任务请求过多资源的机会。而资源和进程的数量对实验结果的倾向影响较小。

1. 实验结果的算法时间、效率、鲁棒性等性能分析。

该算法较为直白，每次循环的次数随机性大，难以预测。若以标准分布来估计之，可以近似认为每次寻找可分配进程符合O(N)时间复杂度，确认是否是安全序列的过程符合O(logN)的时间复杂度。

1. **实验总结**

经过这个实验，我了解掌握了银行家算法的基础。

1. **思考题**
2. 如何设计程序的输入模块才能满足实验要求，请举例说明

\* 资源初始化不得超过一个最大值

\* 程序最大需求不得超过一个最大值

\* 程序已分配资源不得超过最大需求

\* 不得出现负剩余和需求

1. 银行家算法在实现过程中必须注意哪些资源分配细节才能避免死锁？

只能在资源允许的范围内分配资源。

1. **实验数据及源代码（学生必须提交自己设计的程序源代码，并有注释，源代码电子版也一并提交），包括思考题的程序。**

#include <stdio.h>  
#include <stdbool.h>  
#include <stdlib.h>  
#include <time.h>  
  
#define COUNT\_RESOURCE 10 *// total categories of resources*#define COUNT\_PROCESS 10 *// count of processes*#define INITIAL\_CAP 20 *// cap of initial available resources*#define DEMAND\_CAP 20 *// cap of maximum demanded resources*int available[COUNT\_RESOURCE]; *// available resources*int requirement[COUNT\_PROCESS][COUNT\_RESOURCE]; *// maximum demand of every process*int allocated[COUNT\_PROCESS][COUNT\_RESOURCE]; *// allocated of every process*int pending[COUNT\_PROCESS]; *// processes that are not executed*int pending\_count = COUNT\_PROCESS; *// count of pending list*bool can\_satisfy(int pi) {  
 for (int i = 0; i < COUNT\_RESOURCE; ++i) {  
 if (allocated[pi][i] + available[i] < requirement[pi][i]) {  
 return false;  
 }  
 }  
 return true;  
}  
  
void execute\_unchecked(int pi) {  
 int target = pending[pi];  
 printf("---EXECUTION OF P%d---\n", target);  
 for (int i = 0; i < COUNT\_RESOURCE; ++i) {  
 printf("\tR%d", i);  
 }  
 printf("\nBEFORE");  
 for (int i = 0; i < COUNT\_RESOURCE; ++i) {  
 printf("\t%d", available[i]);  
 }  
 printf("\nAFTER");  
 for (int i = 0; i < COUNT\_RESOURCE; ++i) {  
 available[i] += allocated[target][i];  
 }  
 for (int i = 0; i < COUNT\_RESOURCE; ++i) {  
 printf("\t%d", available[i]);  
 }  
 printf("\n");  
 --pending\_count;  
 if (pending\_count != 0) {  
 pending[pi] = pending[pending\_count];  
 }  
}  
  
int main() {  
 *// initialize pending list* for (int i = 0; i < COUNT\_PROCESS; ++i) {  
 pending[i] = i;  
 }  
  
 *// randomize test data* srand48((long) time(NULL));  
 for (int i = 0; i < COUNT\_RESOURCE; ++i) {  
 available[i] = (int) (drand48() \* INITIAL\_CAP);  
 }  
 for (int i = 0; i < COUNT\_PROCESS \* COUNT\_RESOURCE; ++i) {  
 \*(((int \*) requirement) + i) = (int) (drand48() \* DEMAND\_CAP);  
 }  
 for (int i = 0; i < COUNT\_PROCESS \* COUNT\_RESOURCE; ++i) {  
 \*(((int \*) allocated) + i) = (int) (drand48() \* \*((int \*) requirement + i));  
 }  
  
 *// summary* printf("--- INITIALIZATION ---\n\tFREE");  
 for (int i = 0; i < COUNT\_PROCESS; ++i) {  
 printf("\tP%d", i);  
 }  
 printf("\n");  
 for (int i = 0; i < COUNT\_RESOURCE; ++i) {  
 printf("R%d\t%d\t", i, available[i]);  
 for (int j = 0; j < COUNT\_PROCESS; ++j) {  
 printf("%d/%d\t", allocated[j][i], requirement[j][i]);  
 }  
 printf("\n");  
 }  
  
 bool mutated = false;  
 while (pending\_count > 0) {  
 for (int i = 0; i < pending\_count; ++i) {  
 int target = pending[i];  
 if (can\_satisfy(target)) { *// if the resources are adequate* execute\_unchecked(i); *// already checked for safety, execute now* mutated = true;  
 break;  
 }  
 }  
 if (!mutated) { *// Oops, we can't execute anything!* printf("Current state is unsafe!\n");  
 break;  
 }  
 mutated = false;  
 }  
}

**注意： 实验报告文件名 学号-姓名-实验1-班级**

**实验数据与源代码 一个压缩包，名字和实验报告规则一样，需要有一个说明文件解释各个文件是什么文件。**

学委应把实验报告放到一个文件夹中，做成一个压缩包提交；实验数据和源代码，一个同学一个压缩包，形成一个文件夹，压缩后提交