姓名：郭俊杰 学号： 2200604138 班级：22计算机2班

日期： 2024.12.30-2025.01.03 成绩： 指导教师：潘巨龙

**银行家算法系统设计方案**

## 设计方案

#### 概述

银行家算法是由 E.W. Dijkstra 提出的，是一种用于避免死锁的资源分配和管理算法。它基于以下假设：

每个进程在执行时会申请一定数量的资源；系统可以分配资源给进程，但在分配时需要保证系统始终处于安全状态，从而避免死锁的发生；每个进程的资源需求是已知的，且每个进程的资源需求不会超过其最大需求。

本系统设计的目标是：

模拟一个拥有多个进程和多种资源的系统，进行动态资源分配；通过银行家算法确保系统在进行资源分配时不会进入死锁状态；显示资源分配的详细过程，并判断系统是否处于安全状态。

系统要求：

有多个并发进程共享多种资源；能够模拟资源的申请与分配过程；在每次分配前，判断资源分配后系统是否仍处于安全状态；如果存在多个安全路径，打印所有的安全路径；系统能够自动初始化资源数据，并接受用户输入资源请求。

#### 流程图

1.初始化资源状态：  
初始化 Available、Max、Allocation 和 Need 数组。

2.资源申请请求：

·用户输入进程的资源申请。

·判断申请资源是否满足需求。

3.资源分配模拟

·将请求的资源暂时分配给进程。

·更新 Available 和 Allocation 数组。

4.安全性检查

·使用银行家算法检查当前分配后是否安全：

·如果系统处于安全状态，则分配资源，返回“资源分配成功”并保存可行的安全序列。

·如果系统不安全，则回滚分配，返回“资源分配失败”。

5.结束判断

·如果所有进程资源申请都处理完成，则结束程序。

具体流程图见图1：

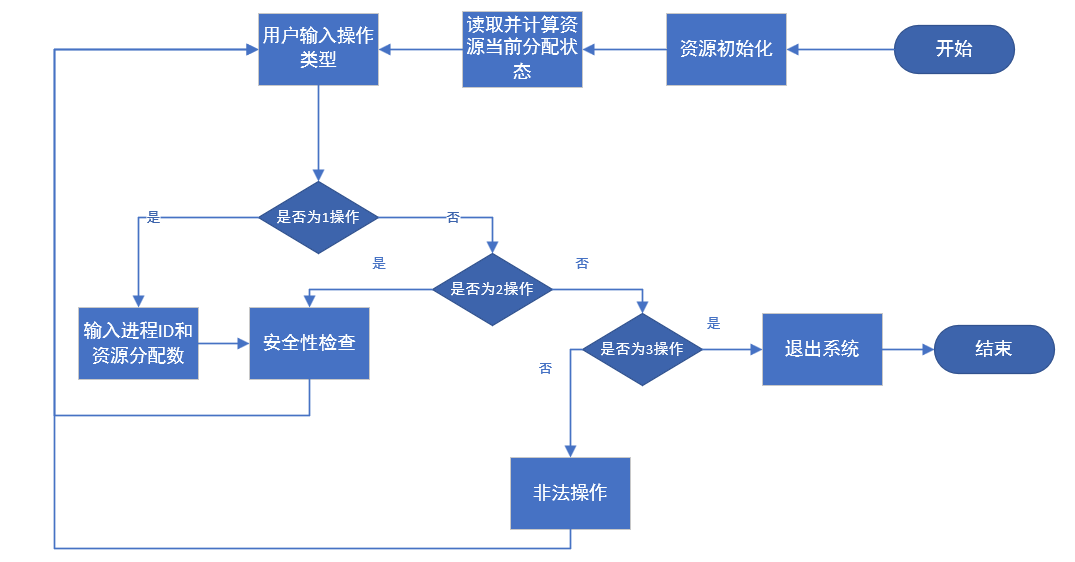


图 1设计流程图

#### 具体实现过程

1.数据结构设计

系统需要处理以下几类数据：

Available：存储系统中每种资源当前的可用数量。

Max：存储每个进程对各资源的最大需求。

Allocation：存储每个进程当前已分配的资源数。

Need：存储每个进程当前剩余需求的资源数，Need[i][j] = Max[i][j] - Allocation[i][j]。

2.算法实现

·资源申请功能

当进程请求资源时，银行家算法会首先进行以下检查：

检查该进程请求的资源数是否超过其最大需求。

检查请求的资源数是否超过系统当前可用的资源数。

如果两个条件都满足，则暂时分配资源，然后进行安全性检查。

·安全性检查功能

安全性检查的核心是判断系统是否存在一个安全序列。银行家算法通过以下步骤进行安全性检查：

创建一个工作数组 res，存储当前系统中可用的资源。

遍历所有进程，找出那些需求小于或等于 res 的进程，并假设这些进程执行完后释放其所有资源。

如果能够找到这样的进程，则将其资源释放回 res 数组。

重复这一过程，直到找到一个完整的安全序列，或者无法找到。

·系统状态判断

根据安全性检查的结果，如果系统能够找到一个安全序列，则表示当前资源分配是安全的；否则，回滚资源分配并返回错误信息。

3.程序流程

初始化资源状态：从文件或硬编码中读取系统资源的初始状态（Available、Max、Allocation）。

资源申请请求：用户输入一个进程 ID 和请求的资源数量。

资源分配模拟：检查资源申请是否合法（是否超过 Need 和 Available）。如果合法，暂时分配资源，更新相关矩阵。

安全性检查：

执行银行家算法判断分配后是否会导致死锁。如果安全，输出所有安全序列；如果不安全，回滚分配。

打印结果：输出当前系统状态，并显示是否安全或错误信息。

4.测试与验证

初始化数据验证：确保从文件读取的资源数据能够正确地初始化系统状态。

资源申请验证：模拟多个进程同时请求资源，验证系统是否能正确判断资源是否分配成功。

安全性检查验证：使用不同的资源请求序列，测试系统是否能够判断出是否存在安全路径。

5.性能优化

资源分配和回滚效率：优化资源分配和回滚的效率，特别是在多次进行资源分配时。

安全性检查优化：优化安全性检查过程中的遍历，减少不必要的计算，提升性能。

并发进程模拟：实现多进程模拟，以检验系统对并发资源请求的处理能力。

## 伪代码

// 矩阵结构体

struct Martix {

int martix[11][11];

int row, col;

};

// 搜索寻找可行的安全序列

void calcSafe(int num) {

if (num <= 0) {

// 找到合法的安全序列

if (cnt == P){

保存安全序列

avi++;

}

return;

}

// 安全序列超过三条，可考虑节约时间结束查找

if (avi >= 3)

return;

for (int i = 1; i <= P; i++) {

if (!ins[i]) {

int can = 1;

for (int j = 1; j <= S && can==1; j++) {

if (res[j] < Need.martix[i][j]) {

can = 0;

}

}

if (can==1) {

// 分配资源，记录序列

que[cnt++]=i;

calcSafe(num - 1);

cnt--;

// 回滚资源及序列

}

}

}

}

int req[10];

// 检查申请资源的合法性

void calcRequest(int num) {

avi = 0;

int is = 1;

// 检查需求是否大于所剩资源

for(int i =1;i<=S;i++){

if (req[i] > res[i]){

printf("需求超出所剩资源\n");

return ;

}

}

// 检查是否超出最大需求

for (int i = 1; i <= S; i++) {

All.martix[num][i] += req[i];

if (All.martix[num][i] > Max.martix[num][i])

is = 0;

}

if (is==0) {

printf("资源超出最大值，请求不合法\n");

// 回滚资源申请

for (int i = 1; i <= S; i++)

All.martix[num][i] -= req[i];

}else{

calcSafe(P);

print(ans)

}

}

int main() {

资源读取和计算

while(1){

// 用户交互

if (申请资源){

Read(req)

calcRequest(p);

}else if (安全性检查){

calcSafe(P);

print(ans)

}else{

break;

}

}

return 0;

}