**中央财经大学**

**实 验 报 告**

**实验项目名称**

**所属课程名称**

**实 验 类 型**

**实 验 日 期**

**班 级**

**学 号**

**姓 名**

**成 绩**

**实验室**

|  |
| --- |
| **实验概述：** |
| **【实验目的及要求】**  **银行家算法的设计与实现**  **【实验原理】**  **银行家算法（Banker's Algorithm）是一种用于避免死锁的算法，它的思想是通过预先分配资源，检查系统是否处于安全状态，从而避免死锁的发生。以下是银行家算法的步骤：**  **1. 定义系统资源：首先，需要确定系统中有哪些资源，以及每种资源的数量。**  **2. 分配资源：当进程请求资源时，银行家算法会检查是否有足够的资源可供分配。如果有，算法会将资源分配给进程，并更新系统中资源的可用数量。**  **3. 检查安全状态：在每次资源分配之后，银行家算法会检查系统是否处于安全状态。安全状态指的是系统能够为所有进程分配所需资源而不会发生死锁。为了检查系统是否处于安全状态，银行家算法会模拟所有进程的资源请求，并检查是否有足够的资源可供分配。如果系统处于安全状态，则可以继续分配资源；否则，需要等待资源可用或者回收已经分配的资源。**  **4. 回收资源：当进程不再需要某些资源时，需要将这些资源归还给系统。银行家算法会更新系统中资源的可用数量，并重新检查系统是否处于安全状态。**  **安全状态检测算法（Safety Detection Algorithm）是银行家算法的核心部分，它用于检查系统是否处于安全状态。以下是安全状态检测算法的步骤：**  **1. 初始化：首先，需要初始化系统资源的可用数量、每个进程已经得到的资源数量以及每个进程还需要的资源数量。**  **2. 查找可分配资源：安全状态检测算法会查找可以分配的资源，即系统中有足够数量的资源可供分配给某个进程。**  **3. 模拟资源分配：对于每个可以分配资源的进程，安全状态检测算法会模拟将资源分配给该进程，并更新系统资源的可用数量以及该进程已经得到的资源数量。**  **4. 检查是否处于安全状态：在模拟资源分配之后，安全状态检测算法会检查系统是否处于安全状态。如果系统处于安全状态，则可以继续分配资源；否则，需要等待资源可用或者回收已经分配的资源。**  **5. 回退模拟：如果系统不处于安全状态，则需要回退模拟的资源分配，并将已经得到的资源数量和还需要的资源数量恢复到模拟之前的状态。**  **6. 重复步骤2-5，直到所有进程都得到了所需的资源或者无法继续分配资源为止。**  **总的来说，银行家算法和安全状态检测算法的思想是通过预先分配资源并检查系统是否处于安全状态，避免死锁的发生。算法的核心是模拟资源分配并检查系统是否处于安全状态。如果系统处于安全状态，则可以继续分配资源；否则，需要等待资源可用或者回收已经分配的资源。**  **【实验环境】（使用的软件）**  **Linux虚拟机，gcc编译器** |
| **实验内容：** |
| **【实验方案设计】**  **流程图：**    **【实验过程】（实验步骤、记录、数据、分析）**  #include<stdio.h> //头文件名,包含输入输出函数等  #include<stdlib.h> //头文件名  #include<time.h> //头文件名  #define PROCESS\_NUMBER 5 //宏定义,定义进程个数为5  #define RESOURCE\_NUMBER 3 //宏定义,资源类个数是3  #define true 1 //宏定义,定义true为1  #define false 0 //宏定义，定义false为0  typedef int bool; //宏定义，定义bool为int类型  int Available[RESOURCE\_NUMBER]={4,5,3}; //系统中三类可利用资源数分别为4，5，3  int Max[PROCESS\_NUMBER][ RESOURCE\_NUMBER]={{8,4,2},{3,3,2},{9,0,2},{2,1,2},{4,3,3}}; //每个进程所需求的资源最大数  int Allocation[PROCESS\_NUMBER][RESOURCE\_NUMBER]={  {0,1,1},  {2,0,0},  {2,0,2},  {2,1,1},  {0,0,2}};  int Need[PROCESS\_NUMBER][ RESOURCE\_NUMBER]; //各进程的当前需求向量  bool compare(int \*a,int \*b,int n) //比较两个一维数组，如果a中各项均大于b，则返回true，否则为false  { int i;  for(i=0;i<n;i++)  if(a[i]<b[i])  return false;  return true;  }  void assign(int \*a,int \*b,int n) //将数组b的值赋给a,n为数组的大小  { int i;  for(i=0;i<n;i++)  a[i]=b[i];  }  void add(int \*a,int \*b,int n) //两个一维数组的加法，各对应项相加  {  int i;  for(i=0;i<n;i++)  a[i]+=b[i];  }  void substract(int \*a,int\*b,int n) //两个一维数组的减法，各对应项相减  {  int i;  for(i=0;i<n;i++)  a[i]-=b[i];  }  void print(int \*a,int n) //打印一维数组  {  int i;  for(i=0;i<n;i++)  printf("%4d",a[i]);  printf("\n");  }  bool issafe(int \*sp) //判断是否是安全状态  {  int i;  int count=0; //记录finish[i]=true 的进程个数为0  int n=0;  int work[RESOURCE\_NUMBER];  bool finish[PROCESS\_NUMBER];  //work=av  assign(work,Available, RESOURCE\_NUMBER);  //初始化标记 finish  for(i=0;i< PROCESS\_NUMBER;i++)  finish[i]=false;  n= PROCESS\_NUMBER; //n为进城的个数  while(n--) //循环最多执行n次  for(i=0;i< PROCESS\_NUMBER;i++)  if(finish[i]==false&&compare(work,Need[i], RESOURCE\_NUMBER)) //判断能否满  //足进程i的要求，work＞＝need[i]是否成立  {  add(work,Allocation[i], RESOURCE\_NUMBER); //分配，待进程完成后再释放  finish[i]=true;  sp[count]=i; //记录安全路径  count++; //记录能够满足的进程数+1  }  if(count>= PROCESS\_NUMBER)  return true;  else  return false;  }  bool request(int pid,int \*r,int n)  {  int i;  int sp[5]; //记录安全路径  if(compare(Need[pid],r,n)==true&&compare(Available,r,n)==true) //如果  //request[i]<=need[i]且request[i]<=available  { //尝试分配资源  substract(Available,r, RESOURCE\_NUMBER);  add(Allocation[pid],r, RESOURCE\_NUMBER);  substract(Need[pid],r, RESOURCE\_NUMBER);  if(issafe(sp)) //判断是否是安全状态  {  printf("Security Path:\n\t");  for(i=0;i< PROCESS\_NUMBER;i++) //打印安全路径  printf("p%d",sp[i]);  printf("\n"); //可以分配  return true;  }  else  { add(Available,r, RESOURCE\_NUMBER); //不分配  substract(Allocation[pid],r, RESOURCE\_NUMBER); //恢复到分配前的状态  add(Need[pid],r, RESOURCE\_NUMBER);  return false;  }    }  else return false;  }  //提示信息  char hint()  { char ch;  printf("\t------------------Operation Hint----------\n");  //按A或者a键自动分配资源  printf("\tA(a)------------Apply For resource automated\n");  //按H或者h键自动分配资源  printf("\tH(h)-------------Apply For resource by human\n");  //按Q或者q键自动分配资源  printf("\tQ(q)-------quit\n");  scanf("%c",&ch);  return ch;  }  //显示系统信息  void init(){  int i;;  int temp[RESOURCE\_NUMBER];  printf("Processs Numbers: %d\n Need and Allocation respectively as follow:\n",PROCESS\_NUMBER);  //显示进程最大资源需求  for(i=0;i<PROCESS\_NUMBER;i++)  { printf("\t process %d max need:",i);  print(Max[i],RESOURCE\_NUMBER);  }  printf("\n");  //计算需求向量：Need[i]=MAX[i]-Alocation[i]  for(i=0;i<PROCESS\_NUMBER;i++){  assign(temp,Max[i],RESOURCE\_NUMBER);  substract(temp,Allocation[i],RESOURCE\_NUMBER);  assign(Need[i],temp,RESOURCE\_NUMBER);  }  //显示进程已分配资源  for(i=0;i<PROCESS\_NUMBER;i++){  printf("\t processs %d allocated resources:",i);  print(Allocation[i],RESOURCE\_NUMBER);  }  //显示系统可用资源  printf("\t available resources:\t");  print(Available,RESOURCE\_NUMBER);  }  //输入  void input(int \*r,int n,int \*id)  { int i;  //提示输入进程号  printf("please input process id(0~ %d):",n-1);  //从键盘输入进程号  scanf("%d",id);  for(i=0;i<n;i++){  printf("\nthe numbers of needed resource %d(int):",i);  scanf("%d",&r[i]);  }  //显示刚才输入的数据  printf("\ndata you inputed: Request[%d](",\*id);  for(i=0;i<n;i++)  printf("%d",r[i]);  printf(")\n");  }  //检查输入  bool check(int id,int \*r,int n)  {  int i;  //判断申请资源数目是否合法  for(i=0;i<n;i++)  if(r[i]<0)  return false;  //判断进程号是否合法  if(id>=PROCESS\_NUMBER)  return false;  else  return true;  }  int main()  {  //进程id号  int id;  //控制字符  char control;  //资源请求向量  int r[3];  //显示开始信息  init();  //随机数初始化  srand((int)time(0));  //主控过程  while(1)  {  //提示  control=hint();  if(control=='a'||control=='A')  {  //随机申请资源  id=rand()%5;  r[0]=rand()%5;  r[1]=rand()%5;  r[2]=rand()%5;  //显示申请信息  printf("\tRequest[%d](%d,%d,%d)\n",id,r[0],r[1],r[2]);  if(request(id,r,RESOURCE\_NUMBER))  printf("Alloc Success!\n");  else  printf("Alloc Failed!\n");  }  else if(control=='h'||control=='H')  {  //输入申请信息  input(r,RESOURCE\_NUMBER,&id);  //检查输入是否合法  if(check(id,r,RESOURCE\_NUMBER)==false)  {  printf("\nInput Error!please reinput!\n");  continue;  }  //换行  if(request(id,r,RESOURCE\_NUMBER))  printf("Request Succeed!\n");  else  printf("Request Fail!\n");  }  else if(control=='q'||control=='Q')  exit(0);  //显示当前系统资源和进程情况  printf("Available Resource\n");  print(Available,RESOURCE\_NUMBER);  //显示资源最大需求  printf("process %d max need\n",id);  print(Max[id],RESOURCE\_NUMBER);  //显示已分配资源情况  printf("process %d allocated resources\n",id);  print(Allocation[id],RESOURCE\_NUMBER);  }  return 0;  }  **【结论】（结果）**    **图一**    **图二**    **图三** |
| **【小结】**  **运行结果：**  **首先，程序会显示系统中每个进程的最大资源需求，已分配资源情况以及系统可用资源情况。**  **然后，程序会提示输入操作指令，可以选择自动申请资源或手动输入申请资源。**  **如果选择自动申请资源，程序会随机选择一个进程，并随机申请一定数量的资源。程序会显示申请的进程号和申请的资源数量，并判断是否能够分配资源。如果能够分配，则会显示安全路径，否则会恢复到申请前的状态。**  **如果选择手动输入申请资源，程序会要求输入进程号和申请的资源数量，然后判断输入是否合法。如果合法，则会尝试分配资源，并显示安全路径或恢复到申请前的状态。**  **每次操作完成后，程序会显示当前系统资源和进程情况，包括可用资源、进程的最大需求、已分配资源情况等。**  **原因分析：**  **该程序实现了银行家算法，用于解决资源分配问题。在程序中，通过比较系统可用资源和进程的最大需求以及已分配资源情况，判断是否能够分配资源。如果能够分配，则会将资源分配给进程，并判断是否是安全状态。如果是安全状态，则会显示安全路径；否则会恢复到分配前的状态。**  **通过该程序的实现，可以更好地理解银行家算法的原理和实现过程，掌握资源分配的方法和技巧。**  **实验体会及收获：**  **通过本次实验，我深刻理解了银行家算法的原理和实现过程，掌握了资源分配的方法和技巧。在实验过程中，我不仅学习了C语言的编程技巧，还提高了自己的问题解决能力和编程能力。**  **在今后的实验中，我将更加注重理论学习和实践操作的结合，努力提高自己的实验能力和实践能力，为将来的工作和学习打好基础。**  **建设性建议：**  **在实验中，我认为可以增加一些交互性的功能，比如可以让用户自己输入进程的最大需求和已分配资源情况，然后再进行资源分配。这样可以更好地模拟实际情况，提高实验的可操作性和实用性。同时，可以增加一些提示信息，帮助用户更好地理解程序的运行过程和结果。** |
| **指导教师评语及成绩：** |
| **评语：**  **成绩： 指导教师签名：**  **批阅日期：** |

实验报告说明

**1．实验项目名称：**要用最简练的语言反映实验的内容。要求与实验指导书中相一致。

**2．实验类型：**一般需说明是验证型实验还是设计型实验，是创新型实验还是综合型实验。

**3．实验目的与要求**：目的要明确，要抓住重点，符合实验指导书中的要求。

**4．实验原理：**简要说明本实验项目所涉及的理论知识。

**5．实验环境**：实验用的软硬件环境（配置）。

**6．实验方案设计（思路、步骤和方法等）**：这是实验报告极其重要的内容。概括整个实验过程。

对于**操作型实验**，要写明依据何种原理、操作方法进行实验，要写明需要经过哪几个步骤来实现其操作。

对于**设计型和综合型实验**，在上述内容基础上还应该画出流程图、设计思路和设计方法，再配以相应的文字说明。

对于**创新型实验**，还应注明其创新点、特色。

**7．实验过程（实验中涉及的记录、数据、分析）：**写明具体上述实验方案的具体实施，包括实验过程中的记录、数据和相应的分析。

**8．结论（结果）：**即根据实验过程中所见到的现象和测得的数据，做出结论。

**9．小结：**对本次实验的心得体会、思考和建议。

**10．指导教师评语及成绩：**指导教师依据学生的实际报告内容，用简练语言给出本次实验报告的评价和价值。

**注意：**

* 实验报告将记入实验成绩；
* 每次实验开始时，交上一次的实验报告，否则将扣除此次实验成绩。