# 实验六

时间显示 ↓



1. **单类资源的银行家算法（找出一个安全序列即可）**

打开文本编辑器，完成work.c ↓

|  |
| --- |
|  |

完整代码如下 ↓

|  |
| --- |
| #include <stdio.h>  #include <stdbool.h>  #define MAX\_PROCESSES 10  int num\_processes;  int available;  int maximum[MAX\_PROCESSES];  int allocation[MAX\_PROCESSES];  int need[MAX\_PROCESSES];  bool is\_safe(int safe\_sequence[]) {  int work = available;  bool finish[MAX\_PROCESSES];  int count = 0;  for (int i = 0; i < num\_processes; i++) {  finish[i] = false;  }  while (count < num\_processes) {  bool found = false;  for (int p = 0; p < num\_processes; p++) {  if (!finish[p]) {  if (need[p] <= work) {  work += allocation[p];  safe\_sequence[count++] = p;  finish[p] = true;  found = true;  }  }  }  if (!found) {  return false; // 系统不在安全状态  }  }  return true; // 系统处于安全状态  }  int main() {  // 输入  num\_processes = 5;  available = 10;  int initial\_maximum[] = {7, 3, 9, 2, 4};  for(int i = 0; i < num\_processes; i++) {  maximum[i] = initial\_maximum[i];  }  int initial\_allocation[] = {0, 2, 3, 2, 0};  for(int i = 0; i < num\_processes; i++) {  allocation[i] = initial\_allocation[i];  need[i] = maximum[i] - allocation[i];  }  int safe\_sequence[MAX\_PROCESSES];  if (is\_safe(safe\_sequence)) {  printf("System is in a safe state.\nSafe sequence: ");  for (int i = 0; i < num\_processes; i++) {  printf("P%d", safe\_sequence[i]);  if (i < num\_processes - 1) {  printf(" -> ");  }  }  printf("\n");  } else {  printf("System is not in a safe state.\n");  }  return 0;  } |

编译文件，并运行↓

|  |
| --- |
|  |

结果分析：

程序处理了多种资源类型的情况，成功找到安全序列，验证了算法的正确性，输出显示系统可以安全地分配资源给各个进程。

1. **多类资源的银行家算法（找出一个安全序列即可）**

打开文本编辑器，完成work1.c ↓

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <stdbool.h>  #define MAX\_PROCESSES 10  #define MAX\_RESOURCES 10  int num\_processes;  int num\_resources;  int available[MAX\_RESOURCES];  int maximum[MAX\_PROCESSES][MAX\_RESOURCES];  int allocation[MAX\_PROCESSES][MAX\_RESOURCES];  int need[MAX\_PROCESSES][MAX\_RESOURCES];  bool is\_safe(int safe\_sequence[]) {  int work[MAX\_RESOURCES];  bool finish[MAX\_PROCESSES];  int count = 0;  // 初始化 work 和 finish  for (int i = 0; i < num\_resources; i++) {  work[i] = available[i];  }  for (int i = 0; i < num\_processes; i++) {  finish[i] = false;  }  while (count < num\_processes) {  bool found = false;  for (int p = 0; p < num\_processes; p++) {  if (!finish[p]) {  bool possible = true;  for (int r = 0; r < num\_resources; r++) {  if (need[p][r] > work[r]) {  possible = false;  break;  }  }  if (possible) {  for (int r = 0; r < num\_resources; r++) {  work[r] += allocation[p][r];  }  safe\_sequence[count++] = p;  finish[p] = true;  found = true;  }  }  }  if (!found) {  return false; // 系统不在安全状态  }  }  return true; // 系统处于安全状态  }  int main() {  // 示例输入（您可以修改）  num\_processes = 5;  num\_resources = 3;  int initial\_available[] = {3, 3, 2};  for(int i = 0; i < num\_resources; i++) {  available[i] = initial\_available[i];  }  int initial\_maximum[5][3] = {  {7, 5, 3},  {3, 2, 2},  {9, 0, 2},  {2, 2, 2},  {4, 3, 3}  };  for(int i = 0; i < num\_processes; i++) {  for(int j = 0; j < num\_resources; j++) {  maximum[i][j] = initial\_maximum[i][j];  }  }  int initial\_allocation[5][3] = {  {0, 1, 0},  {2, 0, 0},  {3, 0, 2},  {2, 1, 1},  {0, 0, 2}  };  for(int i = 0; i < num\_processes; i++) {  for(int j = 0; j < num\_resources; j++) {  allocation[i][j] = initial\_allocation[i][j];  need[i][j] = maximum[i][j] - allocation[i][j];  }  }  int safe\_sequence[MAX\_PROCESSES];  if (is\_safe(safe\_sequence)) {  printf("System is in a safe state.\nSafe sequence: ");  for (int i = 0; i < num\_processes; i++) {  printf("P%d", safe\_sequence[i]);  if (i < num\_processes - 1) {  printf(" -> ");  }  }  printf("\n");  } else {  printf("System is not in a safe state.\n");  }  return 0;  } |

1. **多类资源的银行家算法（找出所有安全序列）**

打开文本编辑器，完成work2.c ↓

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <stdbool.h>  #define MAX\_PROCESSES 10  #define MAX\_RESOURCES 10  int num\_processes;  int num\_resources;  int available[MAX\_RESOURCES];  int maximum[MAX\_PROCESSES][MAX\_RESOURCES];  int allocation[MAX\_PROCESSES][MAX\_RESOURCES];  int need[MAX\_PROCESSES][MAX\_RESOURCES];  void find\_all\_safe\_sequences(int current\_work[], bool current\_finish[], int current\_sequence[], int count) {  if (count == num\_processes) {  // 找到一个安全序列，打印它  printf("Safe sequence: ");  for (int i = 0; i < num\_processes; i++) {  printf("P%d", current\_sequence[i]);  if (i < num\_processes - 1) {  printf(" -> ");  }  }  printf("\n");  return;  }  for (int p = 0; p < num\_processes; p++) {  if (!current\_finish[p]) {  bool possible = true;  for (int r = 0; r < num\_resources; r++) {  if (need[p][r] > current\_work[r]) {  possible = false;  break;  }  }  if (possible) {  // 尝试这个进程  int next\_work[MAX\_RESOURCES];  bool next\_finish[MAX\_PROCESSES];  int next\_sequence[MAX\_PROCESSES];  // 复制当前状态  for(int r = 0; r < num\_resources; r++) next\_work[r] = current\_work[r];  for(int i = 0; i < num\_processes; i++) next\_finish[i] = current\_finish[i];  for(int i = 0; i < count; i++) next\_sequence[i] = current\_sequence[i];  // 更新状态以便进行下一次调用  for (int r = 0; r < num\_resources; r++) {  next\_work[r] += allocation[p][r];  }  next\_finish[p] = true;  next\_sequence[count] = p;  find\_all\_safe\_sequences(next\_work, next\_finish, next\_sequence, count + 1);  // 回溯（状态已复制，因此work和finish数组无需显式回溯）  }  }  }  }  int main() {  // 示例输入（与之前相同）  num\_processes = 5;  num\_resources = 3;  int initial\_available[] = {3, 3, 2};  for(int i = 0; i < num\_resources; i++) {  available[i] = initial\_available[i];  }  int initial\_maximum[5][3] = {  {7, 5, 3},  {3, 2, 2},  {9, 0, 2},  {2, 2, 2},  {4, 3, 3}  };  for(int i = 0; i < num\_processes; i++) {  for(int j = 0; j < num\_resources; j++) {  maximum[i][j] = initial\_maximum[i][j];  }  }  int initial\_allocation[5][3] = {  {0, 1, 0},  {2, 0, 0},  {3, 0, 2},  {2, 1, 1},  {0, 0, 2}  };  for(int i = 0; i < num\_processes; i++) {  for(int j = 0; j < num\_resources; j++) {  allocation[i][j] = initial\_allocation[i][j];  need[i][j] = maximum[i][j] - allocation[i][j];  }  }  int current\_sequence[MAX\_PROCESSES];  bool current\_finish[MAX\_PROCESSES] = {false};  int current\_work[MAX\_RESOURCES];  for(int i = 0; i < num\_resources; i++) current\_work[i] = available[i];  printf("Finding all safe sequences:\n");  find\_all\_safe\_sequences(current\_work, current\_finish, current\_sequence, 0);  return 0;  } |