



### A. MỤC TIÊU

Sau khi hoàn thành bài thực hành này, sinh viên sẽ đạt được các kỹ năng sau:

- Viết được chương trình minh họa việc thực thi của các giải thuật quản lý tắc nghẽn như: Tránh tắc nghẽn, dò tìm tắc nghẽn, ngăn chặn tắc nghẽn bằng ngôn ngữ C.
- Áp dụng được các ngôn ngữ lập trình khác nhau để viết minh họa giải thuật.
- Hình thành tư duy xử lý bài toán lập trình.

### B. DỤNG CỤ - THIẾT BỊ THỰC HÀNH CHO MỘT SINH VIÊN:

STT	Chủng loại – quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	Computer	1	Bộ	

### C. NỘI DUNG THỰC HÀNH

#### 1. Tóm tắt lý thuyết

Bài thực hành liên quan đến các vấn đề lý thuyết về giải quyết tắc nghẽn trong hệ điều hành, cụ thể như sau:

- Giải thuật phát hiện tắc nghẽn
- Giải thuật tránh tắc nghẽn
- Giải thuật ngăn chặn tắc nghẽn

#### 2. Bài thực hành trên lớp

##### 2.1. Bài tập mẫu

**Yêu cầu:** Viết chương trình C minh họa việc thực thi giải thuật Banker tránh tắc nghẽn?

### Chương trình mẫu:

```
1  /*
2  CHƯƠNG TRÌNH MINH HỌA GIẢI THUẬT BANKER
3  MÔN: THỰC HÀNH HỆ ĐIỀU HÀNH
4  BỘ MÔN: MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG
5  KHOA: CNTT - ĐH CNTP
6  */
7  #include <stdio.h>
8  int main()
9  {
10     // P0, P1, P2, P3, P4 Tên của các tiến trình minh họa
11
12     int n, m, i, j, k;
13     n = 5; // Số tiến trình
14     m = 3; // Số tài nguyên
15     int alloc[5][3] = { { 0, 1, 0 }, // P0 // Ma trận Allocation
16                         { 2, 0, 0 }, // P1
17                         { 3, 0, 2 }, // P2
18                         { 2, 1, 1 }, // P3
19                         { 0, 0, 2 } }; // P4
20
21     int max[5][3] = { { 7, 5, 3 }, // P0 // Ma trận MAX
22                      { 3, 2, 2 }, // P1
23                      { 9, 0, 2 }, // P2
24                      { 2, 2, 2 }, // P3
25                      { 4, 3, 3 } }; // P4
26
27     int avail[3] = { 3, 3, 2 }; // Tài nguyên sẵn hiện tại có của hệ thống
28
29     int f[n], ans[n], ind = 0;
30     for (k = 0; k < n; k++) {
31         f[k] = 0;
32     }
33     int need[n][m];
34     for (i = 0; i < n; i++) {
35         for (j = 0; j < m; j++)
36             need[i][j] = max[i][j] - alloc[i][j];
37     }
38     int y = 0;
39     for (k = 0; k < 5; k++) {
40         for (i = 0; i < n; i++) {
41             if (f[i] == 0) {
42
43                 int flag = 0;
44                 for (j = 0; j < m; j++) {
45                     if (need[i][j] > avail[j]){
46                         flag = 1;
47                         break;
48                     }
49                 }
50
51                 if (flag == 0) {
52                     ans[ind++] = i;
53                     for (y = 0; y < m; y++)
54                         avail[y] += alloc[i][y];
55                     f[i] = 1;
56                 }
57             }
58         }
59     }
60
61     printf("Chuỗi an toàn là:\n");
62     for (i = 0; i < n - 1; i++)
63         printf(" P%d ->", ans[i]);
64     printf(" P%d", ans[n - 1]);
65
66     return (0);
67
68
69 }
70
```

Kết quả chạy chương trình:

Chuỗi an toàn là: P1 -> P3 -> P4 -> P0 -> P2

## 2.2. Bài tập luyện tập

Bài tập 1. **Viết chương trình C minh họa việc thực thi giải thuật Banker dò tìm tắc nghẽn?**

Bài tập 2. **Viết chương trình C minh họa việc thực thi giải thuật Banker ngăn chặn tắc nghẽn?**

## 3. Bài thực hành ở nhà

Hãy thực hiện lại các bài tập sau bằng Windows Form. Yêu cầu: Nộp lại source code vào thư mục trên kênh học tập Google Classroom.

- Bài tập 1. **Viết chương trình C minh họa việc thực thi giải thuật Banker?**

Ví dụ giao diện Giải thuật

The screenshot shows the 'bankersAlgo' application window. It contains several input fields and buttons for configuring the Banker's Algorithm simulation.

**Allocation:** Process P4, Resource 1: 0, Resource 2: 0, Resource 3: 2. Button: Add Process.

**Max matrix:** Resource 1: 4, Resource 2: 3, Resource 3: 3. Button: Add Process.

**Available:** Resource 1: 3, Resource 2: 3, Resource 3: 2. Button: Calculate.

**Allocation Table:**

Process	A	B	C
P0	0	1	0
P1	2	0	0
P2	3	0	2
P3	2	1	1
P4	0	0	2

**Max Allocation:**

	A	B	C
7	5	3	
3	2	2	
9	0	2	
2	2	2	
4	3	3	

**Need Matrix:**

	A	B	C
7	4	3	
1	2	2	
6	0	0	
0	1	1	
4	3	1	

**Available Matrix:**

	A	B	C
5	3	2	
7	4	3	
7	4	5	
7	5	5	
10	5	7	

**Safe Sequence:**

P1 Allocated  
P3 Allocated  
P4 Allocated  
P0 Allocated  
P2 Allocated

**Safely allocated**