COMP 322/L—Introduction to Operating Systems and System Architecture
Assignment #3—Banker's Algorithm

Objective:

To implement resource requests and resource releases and demonstrate deadlock avoidance using the Banker's algorithm.

Specification:

The program simulates resource requests/releases to/from processes and demonstrates deadlock avoidance with the Banker's algorithm. A menu controls the operations, and each choice calls the appropriate procedure, where the choices are:

1) Enter claim graph
2) Request resource
3) Release resource
4) Determine safe sequence
5) Quit program and free memory

Assignment:

- The program uses a claim graph consisting of processes, multi-unit resources, request edges, allocation edges, and claim edges to represent the state of allocated resources to processes.
- The graph can be represented by a set of arrays:
  - ***Resource vector:*** an *m*-element vector, where *m* is the number of resources and each entry **resource[*j*]** records the total number of units of resource *j*
  - ***Available vector:*** an *m*-element vector, where *m* is the number of resources and each entry **available[*j*]** records the number of units of resource *j* that are available
  - ***Max claims array:*** an *n*x*m*-element array, where *m* is the number of resources and *n* is the number of processes, and each entry **maxclaim[*i*][*j*]** records the maximum number of units of resource *j* that process *i* may claim
  - ***Allocated array:*** an *n*x*m*-element array, where *m* is the number of resources and *n* is the number of processes, and each entry **allocated[*i*][*j*]** records the number of units of resource *j* that process *i* has been allocated
  - ***Need array:*** an *n*x*m* array, where *m* is the number of resources and *n* is the number of processes, and each entry **need[*i*][*j*]** records the number of units of resource *j* that process *i* may need in the future

What NOT to do:

- Do NOT modify the choice values (1,2,3,4,5) or input characters and then try to convert them to integers--the test script used for grading your assignment will not work correctly.
- Do NOT turn in an alternate version of the assignment downloaded from the Internet (coursehero, chegg, reddit, github, ChatGPT, etc.) or submitted from you or another student from a previous semester—the test cases from this semester will not work on a previous semester's assignment.
- Do NOT turn in your assignment coded in another programming language (C++, C#, Java).

What to turn in:

- The source code as a C file uploaded to Canvas by the deadline of 11:59pm PST. Please check the syllabus for the late submission policy.—note 1-minute late counts as a day late, 1 day and 1 minute late counts as 2 days late, etc.)
- As a note, even though your code may compile on a compiler you have installed on your computer, I do not have access to your computer. I will be using the following free online compiler for testing, so make sure your code compiles with the following online C compiler before submitting: https://www.onlinegdb.com/online_c_compiler If it does not compile with the above compiler, the default grade is 0 points since I cannot run it.

Sample output

```
Banker's Algorithm
------------------
1) Enter claim graph
2) Request resource
3) Release resource
4) Determine safe sequence
5) Quit program

Enter selection: 1
Enter number of resources: 3
Enter number of units for resources (r0 to r2): 10 5 7
Enter number of processes: 5
Enter maximum number of units process p0 will claim from each resource (r0 to r2): 7 5 3
Enter maximum number of units process p1 will claim from each resource (r0 to r2): 3 2 2
Enter maximum number of units process p2 will claim from each resource (r0 to r2): 9 0 2
Enter maximum number of units process p3 will claim from each resource (r0 to r2): 2 2 2
Enter maximum number of units process p4 will claim from each resource (r0 to r2): 4 3 3
Enter number of units of each resource (r0 to r2) currently allocated to process p0: 0 1 0
Enter number of units of each resource (r0 to r2) currently allocated to process p1: 2 0 0
Enter number of units of each resource (r0 to r2) currently allocated to process p2: 2 0 2
Enter number of units of each resource (r0 to r2) currently allocated to process p3: 2 1 1
Enter number of units of each resource (r0 to r2) currently allocated to process p4: 0 1 2

Resources:
        r0      r1      r2
        10      5       7

Available:
        r0      r1      r2
        4       2       2

Max Claim:
        r0      r1      r2
p0      7       5       3
p1      3       2       2
p2      9       0       2
p3      2       2       2
p4      4       3       3

Allocated:
        r0      r1      r2
p0      0       1       0
p1      2       0       0
p2      2       0       2
p3      2       1       1
p4      0       1       2

Need:
        r0      r1      r2
p0      7       4       3
p1      1       2       2
p2      7       0       0
p3      0       1       1
p4      4       2       1

Banker's Algorithm
------------------
1) Enter claim graph
2) Request resource
3) Release resource
4) Determine safe sequence
5) Quit program

Enter selection: 2
Enter requesting process: p2
Enter requested resource: r0
Enter number of units process p2 is requesting from resource r0: 1

Available:
        r0      r1      r2
        3       2       2

Allocated:
        r0      r1      r2
p0      0       1       0
p1      2       0       0
```

```
p2      3       0       2
p3      2       1       1
p4      0       1       2

Need:
        r0      r1      r2
p0      7       4       3
p1      1       2       2
p2      6       0       0
p3      0       1       1
p4      4       2       1

Banker's Algorithm
------------------
1) Enter claim graph
2) Request resource
3) Release resource
4) Determine safe sequence
5) Quit program

Enter selection: 3
Enter releasing processor: p4
Enter released resource: r1
Enter number of units process p4 is releasing from resource r1: 1

Available:
        r0      r1      r2
        3       3       2

Allocated:
        r0      r1      r2
p0      0       1       0
p1      2       0       0
p2      3       0       2
p3      2       1       1
p4      0       0       2

Need:
        r0      r1      r2
p0      7       4       3
p1      1       2       2
p2      6       0       0
p3      0       1       1
p4      4       3       1

Banker's Algorithm
------------------
1) Enter claim graph
2) Request resource
3) Release resource
4) Determine safe sequence
5) Quit program

Enter selection: 4
Comparing: < 7 4 3 > <= < 3 3 2 > : Process p0 cannot be sequenced
Comparing: < 1 2 2 > <= < 3 3 2 > : Process p1 can be sequenced
Comparing: < 6 0 0 > <= < 5 3 2 > : Process p2 cannot be sequenced
Comparing: < 0 1 1 > <= < 5 3 2 > : Process p3 can be sequenced
Comparing: < 4 3 1 > <= < 7 4 3 > : Process p4 can be sequenced
Comparing: < 7 4 3 > <= < 7 4 5 > : Process p0 can be sequenced
Comparing: < 6 0 0 > <= < 7 5 5 > : Process p2 can be sequenced
Safe sequence of processes: p1  p3  p4  p0  p2

Banker's Algorithm
------------------
1) Enter claim graph
2) Request resource
3) Release resource
4) Determine safe sequence
5) Quit program

Enter selection: 5
Quitting program...
```