Project TA - Peter CHUNG (cspeter@cse.ust.hk): Handling questions before the project deadline
Grader TA - CHEN, Wei (wchenbt@cse.ust.hk): Collecting late submissions, grading and handling appeals

COMP3511 Operating System
Fall 2020

Project Assignment 2 (PA2) – Banker's algorithm (Safety)

06 Nov 2020 (Friday) 02 Dec 2020 (Wednesday) 23:59

Introduction

The aim of this project is to help students understand **deadlock detection** in an operating system. Upon completion of the project, students should be able to understand and implement a fundamental deadlock detection algorithm: banker's algorithm

Overview

You need to implement a deadlock detection program using banker's algorithm. The program is named as `banker_safety`. Here is a sample usage:

```
$> ./banker_safety < input.txt > output.txt
```

`$>` represents the shell prompt.
< means input redirection.
> means output redirection.

Getting Started

`banker_safety_skeleton.c` is provided. You should rename the file as `banker_safety.c`

In the given base code, necessary input handling is completed. Suitable constants and variables for the banker's algorithm are given. Students can be focused on the implementation of the banker's algorithm.

Development Environment

CS Lab 2 is the development environment. Please use one of the following machines (`csl2wk`**`XX`**`.cse.ust.hk`), where **`XX`**=`01…40`. The grader TA will use the same platform to grade all submissions.

In other words, *"my program works on my own laptop/desktop computer, but not in one of the CS Lab 2 machines"* is an invalid appeal reason. Please test your program on our development environment (not on your own desktop/laptop) thoughtfully

The Input and Output Format

The input handling is given in the base code.

The output format consists of 3 regions:

- The first region is the banker's algorithm input values. Because the input handling is provided in the base code, you don't need to implement this part
- The second region shows the intermediate steps of the banker's algorithm. In each step, you should show the values of the work vector and the finish vector
- The third region shows the result of the banker's algorithm. You should print out whether the system is safe and then print out the execution sequence

Here is a sample input (input1.txt):

```
# The input file for banker's algorithm
# Empty lines and lines starting with '#' are ignored

# file format:
# Line 1 describes the algorithm: Safety or Resource-Request
# Line 2 is the number of resource type (m)
# Line 3 is the number of process (n)
# Line 4 is the available vector (m non-negative integers)
# Line 5 to 5+n-1: Allocation matrix, each line has m non-negative integers
# Line 5+n to 5+2n-1: Max matrix, each line has m non-negative integers

# Algorithm
Safety

# Number of resource type
4

# Number of process
5

# Available vector
2 1 2 0

# Allocation matrix
0 0 1 2
2 0 0 0
0 0 3 4
2 3 5 4
0 3 3 2

# Max matrix
0 0 3 2
2 7 5 0
6 6 5 6
4 3 5 6
0 6 5 2
```

Here is the command to run the sample input and the corresponding sample output:

```
$ ./banker_safety < input1.txt
=== Banker's algorithm input values ===
algorithm = Safety
m = 4
n = 5
work = [2,1,2,0]
alloc = [
0,0,1,2
2,0,0,0
0,0,3,4
2,3,5,4
0,3,3,2
```

```
]
max = [
0,0,3,2
2,7,5,0
6,6,5,6
4,3,5,6
0,6,5,2
]
need = [
0,0,2,0
0,7,5,0
6,6,2,2
2,0,0,2
0,3,2,0
]
finish = [0,0,0,0,0]
=== Banker's algorithm execution ===
=== Step 0 ===
work = [2,1,2,0]
finish = [0,0,0,0,0]
=== Step 1 ===
work = [2,1,3,2]
finish = [1,0,0,0,0]
=== Step 2 ===
work = [4,4,8,6]
finish = [1,0,0,1,0]
=== Step 3 ===
work = [4,7,11,8]
finish = [1,0,0,1,1]
=== Step 4 ===
work = [6,7,11,8]
finish = [1,1,0,1,1]
=== Step 5 ===
work = [6,7,14,12]
finish = [1,1,1,1,1]
=== Banker's algorithm result ===
The system is safe
seq = [0,3,4,1,2]
```

Avoiding multiple solutions in the Banker's algorithm

Banker's algorithm may have more than one possible solution. It is because we find an index `i` such that:

- `finish[i]` is equal to false
- `need_i <= work`

To avoid having more than one possible solution, in this project assignment, we always choose **the smallest index that satisfies the conditions**. For example, if both `i=1` and `i=2` satisfy the conditions, choose `i=1` because 1 is less than 2. Please check the given sample input/output carefully.

Compilation

The following command can be used to compile the program

```
$> gcc -std=c99 -o banker_safety banker_safety.c
```

The option `c99` is used to provide a more flexible coding style (e.g. you can define an integer variable anywhere within a function)

Sample test cases

Several test cases are provided. We don't have other hidden test cases.

The grader TA will probably write a grading script to mark the test cases. Please use the Linux `diff` command to compare your output with the sample output. For example:

```
$> diff --side-by-side your-outX.txt  sample-outX.txt
```

An extra option `--suppress-common-lines` can be added if you are not interested in the common lines. If both text files are the same, adding `--suppress-common-lines` will print nothing on the screen.

Sample Executable

The sample executable (runnable in a CS Lab 2 machine) is provided for reference. After the file is downloaded, you need to add an execution permission bit to the file. For example:

```
$> chmod u+x banker_safety
```

Marking Scheme

1. (20%) Explanation of Banker's algorithm (Safety). You should use point form to explain how you implement this function. Please write the explanation near the top of the source code file (*note: a nice template is already provided near the top*) so that the grader TA can grade your explanation part easily
2. (80%) Correctness of the 8 pairs of test cases (i.e. both input and output files are provided). We don't have hidden test cases. Please use `diff` command to check the output files carefully. Please double-check your program and make sure that your code can be executed in a CS Lab 2 machine

Submission

File to submit: **`banker_safety.c`**

Please check carefully you submit the correct file. In the past, some students submitted an executable file instead of the code file, and we cannot grade an executable file.

You are not required to submit other files such as the input test cases.

You need to submit the file on/before the due date.