Project TA - Peter CHUNG (cspeter@cse.ust.hk): Handling questions before the project deadline
Grader TA – Peter CHUNG (cspeter@cse.ust.hk): Grading and appeal handling after the project deadline

COMP3511 Operating System Fall 2021

PA2: Banker's algorithm (2 variations + 2 strategies)

29-Oct-2021 (Friday) 19-Nov-2021 (Friday) 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 system program that simulates Banker's algorithm. The program is named as `banker`. Here is a sample usage of `banker`:

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

`$>` represents the shell prompt.
`<` means input redirection. `>` means output redirection. Thus, you can easily use the given test cases to test your program and use the `diff` command to compare the output files.

Getting Started

`banker_skeleton.c` is provided. You should rename the file as `banker.c`

You can add new constants, variables, and helper functions

Necessary header files are included. You should not add extra header files.

Assumptions

- There are at most 10 different types of resources
- There are at most 10 different processes

Some print-related constants and helper functions are provided in the starter code, for example, `print_vec` and `print_mat`. They are useful to print out the vector and matrix values

Students needs to parse the input and then implement 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

Input Format

All values are either integers or strings
You can assume that all values are valid

- For example, `num_process` must be a positive integer less than or equal to 10, and so on…

Empty lines are ignored
Lines starting with # are ignored

- Without these comment lines, it is very hard to understand the input files

Format of constant:

- name = <value>

Format of vector:

- name = <values of the vector>

Format of matrix:

- name =
- followed by lines storing the matrix values

Note:

- Resource-Request has 2 extra input values

A sample input (please check the other input test cases by yourself):

```
# COMP3511 PA2 (Fall 2021)
# The input file for banker's algorithm
# Empty lines and lines starting with '#' are ignored

algorithm = Resource-Request
strategy = smallest-index
num_resource = 5
num_process = 5
available_vector = 2 3 1 4 3
allocation_matrix =
3 0 0 2 3
1 0 0 0 1
1 0 3 3 1
0 1 0 0 0
2 3 3 0 0
max_matrix =
5 0 1 4 7
2 5 1 2 2
2 3 3 7 1
9 3 5 7 3
2 6 5 8 0
request_process_id = 0
request_vector = 1 0 1 0 1
```

Output Format

- It is too lengthy to post the output files here, please refer to the output text files
- The output consists of 3 regions:
  - The first region prints out the parsed input values
  - The second region prints the execution steps
    - Two error conditions of the Resource-Request should be handled in the second region. Please check the lecture notes about the error conditions of Resource-Request
  - The third region prints the result of the banker's algorithm

Variations of Banker's Algorithm

In the lecture, we discussed 2 variations: Safety and Resource-Request
In the project, you are required to implement both variations

Strategies to Handle Multiple Solutions

Banker's algorithm may have more than one possible solution. In the standard banker's algorithm, we find any index i such that:

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

It is very hard to grade the programming assignment if we have more than one solution. Thus, to avoid having more than one possible solution, we are going to implement 2 possible strategies:

- smallest-index
  - If we have more than one choice, pick the smallest index that satisfies the conditions
  - For example, if both i=1 and i=2 satisfy the conditions, choose i=1
- largest-index
  - If we have more than one choice, pick the largest index that satisfies the conditions
  - For example, if both i=1 and i=2 satisfy the conditions, choose i=2

Compilation

The following command can be used to compile the program

```
$> gcc -std=c99 -o banker banker.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 (both input files and output files) 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
```

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
```

Marking Scheme

1. (20%) Explanation of Banker's algorithm. You should use point form to explain how you implement this function. To speed up the grading process, please write the comment lines near the top
2. (80%) Correctness of the 8 provided test cases. Again, we don't have other hidden test cases, but we will check the source codes to avoid students hard coding the test cases in their programs. Make sure you use the Linux diff command to check the output format.

***Plagiarism**: Both parties (i.e., students providing the codes and students copying the codes) will receive 0 marks. Near the end of the semester, a plagiarism detection software (MOSS) will be used to identify cheating cases. MOSS was developed by a research team in Stanford. The system is quite robust to detect simple refactoring tricks (e.g., renaming variables, inserting dummy variables, refactoring if-statements /loops, etc.). **DON'T** do any cheating!*

Submission

File to submit:

`banker.c`

Please check carefully you submit the correct file.

In the past semesters, some students submitted the executable file instead of the source file. Zero marks will be given as the grader cannot grade the executable file.

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

Late Submission

For late submission, please submit it via email to the grader TA.
There is a 10% deduction, and only 1 day late is allowed
(Reference: Chapter 1 of the lecture notes)