LE/EECS3221 M - Operating System Fundamentals (Winter 2021-2022)

Assignment 2: Banker's algorithm

Opened: Monday, 7 March 2022, 12:00 AM
Due: Monday, 21 March 2022, 11:59 PM

This assignment is adopted from the lab designed by Prof Allan Gottlieb

The goal of this assignment is to do resource allocation using both an optimistic resource manager and the banker's algorithm of Dijkstra. The optimistic resource manager is simple: Satisfy a request if possible, if not make the task wait; when a release occurs, try to satisfy pending requests in a FIFO manner.

Your program takes one command line argument, the name of the file containing the input. After reading (all) the input, the program performs two simulations: one with the *optimistic manager* and one with the *banker*. Output is written to stdout (the screen).

Example input files are attached below, together with the expected output (which your program is required to match).

To help in debugging there are also cycle by cycle results included for a few of the examples in the "detailed" files. You are **NOT** required to produce these "detailed" outputs; they are for your benefit only. TAs will test your program on additional input as well.

The input begins with two values **T**, the number of tasks, and **R**, the number of resource types, followed by **R** additional values, the number of units present of each resource type. (If you set "arbitrary limits" on say **T** or **R**, you must document this in your readme, check that the input satisfies the limits, print an error if it does not, and set the limits high enough so that the provided inputs all pass.) Then come multiple inputs, each representing the next activity of a specific task. The possible activities are *initiate*, *request*, *compute*, *release*, and *terminate*. Time is measured in fixed units called cycles and, for simplicity, no fractional cycles are used. The manager can process one activity (*initiate*, *request*, or *release*) for each task in one cycle. However, the terminate activity does **not** require a cycle.

To ease the programming, I have specified all activities to have the same format, namely a string followed by three unsigned integers.

The *initiate* activity, which must precede all others for that task, is written

```
initiate task-number resource-type initial-claim
```

(The optimistic manager ignores the claim.) If there are R resource types, there are R initiate activities for each task, each requiring one cycle.

The request and release activities are written

```
request task-number resource-type number-requested
release task-number resource-type number-released
```

The compute activity is written:

```
compute task-number number-of-cycles unused
```

This activity means that for the next several cycles the process is computing and will make no requests or releases. The process retains its current resources during the computation. The fourth value is not used; I include it so that all activities have the same format.

Finally the terminate operation, which does **not** require a cycle is written

```
terminate task-number unused unused
```

The last two values are not used; I include them so that all activities have the same format.

Commenting your program

You must include enough high-level comments in your program so that a reader (e.g., the grader) who is expert in the programming language you use and knowledgeable about resource management can understand the basic operation of your program. For example, you should make clear when you are checking for safety, when you are checking for deadlock, when you are releasing a previously blocked task, etc. You must also supply comments for your major data structures.

Each function/procedure/method you write must have an introductory comment stating what the procedure does and how it does it (at a high level).

Note that these **required** comments will form a (small) portion of the grade.

Input format

As in Assignment 1, we use free form input. Hence a single activity may span several lines and (parts of) several activities may be on one line. For this assignment, I advise you to read all the data before processing. The data for each task occurs in order, but the tasks themselves may be interspersed.

Running your program

Your program must read its input from a file, whose name is given as a command line argument. The format of the input is described above; sample inputs are attached..

Output

At the end of the run, print, for each task, the time taken, the waiting time, and the percentage of time spent waiting. Also print the total time for all tasks, the total waiting time, and the overall percentage of time spent waiting.

See the attached sample outputs and match the format.

Error Checks

When implementing banker's algorithm, if a task's initial claim exceeds the resources present or if, during execution, a task's requests exceed its claims, print an informative message, abort the task, and release all its resources. This does not apply to the optimistic allocator since it does not have the concept of initial claim.

Detecting Deadlocks

Deadlock cannot occur for banker's algorithm, but can for the optimistic resource manager. If deadlock is detected, print a message and abort the lowest numbered deadlocked task after releasing all its resources. If deadlock remains, print another message and abort the next lowest numbered deadlocked task, etc.

We learned sophisticated algorithms for detecting deadlock. You are **not** expected to implement one of these. Instead you should use the following simple algorithm that detects a deadlock when all non-terminated tasks have outstanding requests that the manager cannot satisfy. Note that, if a deadlock actually occurs during cycle n, you may not detect it until much later since there may be non-deadlocked processes running. If you detect the deadlock at cycle k, you abort the task(s) at cycle k and hence its/their resources become available at cycle k+1. This simple deadlock detection algorithm is not used in practice.

Important Note

Items returned at time *n* are not available until time *n+1*, For example, if task 1 returns 10 units during cycle 6-7 and task 3 requests 10 units during that same cycle, the 10 units returned by task 1 are **not** available for the optimistic manager to give out and, when the banker makes its safety check, these 10 units are **not** considered available. They become available at time 7 for use in cycle 7-8.

Example

The input for the first run is:

```
2 1 4
initiate 1 1 4
request 1 1 1
release 1 1 1
terminate 1 0 0
initiate 2 1 4
request 2 1 1
release 2 1 1
terminate 2 0 0
```

- The first line asserts that this run has 2 tasks and 1 resource type with 4 units.
- The next line indicates that the run begins (at cycle 0-1, the cycle starting at 0 and ending at 1) with task 1 claiming (all) 4 units of resource 1. Further down on line 6 we see that task 2 also claims 4 units of resource 1 during cycle 0-1.
- From lines 3 and 7 we learn that each task requests a unit during cycle 1-2 and returns that unit during the next cycle after the request is granted. For the optimistic manager, the request is granted at 2 (the end of cycle 1-2) and the resource is returned during 2-3.
- Input 1 does not use the compute activity. For the optimistic manager each task terminates at time 3.

See additional sample IO files attached below.

What to submit

You need to submit the following a zip file (named: **firstname_lastname_yorkid.zip**) that includes the following required components :

- Individual C implementation of the scheduler, named **banker.c** (can also include other dependency files.
- **Each file** should include the following header:
  /**
  * Full Name:
  * Course ID:
  * Description: *a very brief description of what is implemented (no more than 4 lines*)
  */
- You must include enough high-level comments in your program so that a reader (e.g., the grader/TA) who is expert in C and knowledgeable about the resource allocations algorithms can understand the basic operation of your program. See more details above in the spec.
- Your code should be properly formatted (use can use online services (e.g., *codebeautify*) this one to achieve this)
- The output should be written to stdout (the screen).
- A **README.txt** file describing each included file and dependencies.
- A **Makefile** which will be used to compile and run your program
- You code should terminate within a reasonable amount of time. If you are not sure, please ask.
- **Note: TAs are instructed to deduct points for programs that don't follow the above guidelines and require special adjustment to test (e.g., due to non-trivial compilation and execution, non-conforming to IO formats, etc).**

All submissions will go through a plagiarism detection check. Student who are found to have engaged in academic dishonesty will be referred to the Dean's office for disciplinary action which can result, at minimum, in receiving a FAIL grade in the course, as well as suspension or expulsion from the University.

SampleIO.zip 6 March 2022, 9:55 AM

Submission status

Attempt number	This is attempt 1.
Submission status	No attempt
Grading status	Not marked
Time remaining	14 days 12 hours
Last modified	-
Submission comments	Comments (0)

Add submission

You have not made a submission yet.

◄ Assignment 1: Scheduling

Jump to...

Course Announcements ►