Programming Project 4

Threads

CS 441/541 – Fall 2020

Project Available		Oct. 26
Component	**Points**	**Due Date (at 11:59 pm)**
Proper use of Pthreads	4	
Proper synchronization	4	
Documentation & Testing	2	
Total	10	Nov. 6
Submission	Individual	Bitbucket PR

Objectives

In this project, you'll implement the **dining philosophers** problem as we talked about in class as a way to learn how to use Pthreads and semaphores.

Packaging & handing in your project

A template project is available on BitBucket. You ***MUST*** use the provided semaphore library as it works more reliably on various operating systems.

In order to form a group, use the appropriate tools in Canvas to create your group. If you have any trouble, let the instructor know.

You will submit your code as a pull request (PR) and set me (ssfoley) as a reviewer. In order to do this, be sure to follow our usual workflow for the class:

- **Fork** the template code.
- Give me **write permissions** on the repo. If you are working with a partner, give your teammate write access now, too.
- Create a **branch** called `dev` where you will do your work.
- Clone your dev branch and do the work.
- When you are ready to turn it in, create a **pull request from dev to master** on your fork. Please **make me a reviewer on the PR** so I can easily find your work.

1 The Dining Philosophers Problem

In this part of the assignment you will be implementing **two *solutions*** to the Dining Philosophers Problem as presented in the Little Book of Semaphores.

`http://greenteapress.com/semaphores/LittleBookOfSemaphores.pdf`

The first solution (4.4 - page 87) does not handle deadlock.
The second is Tanenbaum's solution (4.4.5 - page 97) and is deadlock free.

1.1 Requirements

You will need to incorporate the following design points (along with the general design points) in both of your implementations for this part of the assignment.

- You may **only use semaphores** as your synchronization primitive.
- Threads should sleep for a random amount of time between 0 and 5000 microseconds when they are "Eating" and when they are "Thinking".
- Your program should take two (2) command line parameters.
  1. Time to run in seconds. **Required argument**.
     This is the length of time that your application will run before terminating itself.
     The user must specify a positive integer greater than 0.
  2. Number of philosophers at the table. **Optional, default is 5**.
     The user should be able to specify any positive integer greater than 1.
- You will need to print the state of the philosopher whenever their state changes to "Thinking", "Eating", and "Done Eating". Additionally, you will need to assign each philosopher a unique integer identifier which should be printed each time their state changes.
- Before the application exits you must print, for each philosopher, the number of times they were in the eating and thinking states.
- You are expected to turn in **two different source files** one for the deadlock (`diner-v1`) implementation and one for the deadlock free (`diner-v2`) implementation.

1.2 Special Section - Writing about your solution

In a ***special section*** within your documentation you must answer the following questions:

- How can you determine when one or more philosophers are deadlocked in your application?
- How can you determine when one or more philosophers are starving in your application?
- Did deadlock occur every time you ran the deadlock prone version of the *solution* (`diner-v1`)?
- What happens when you change the number of philosophers (in both solutions) to 2? or 4? or 7? or 100?
- While running the Tanenbaum solution with 5 philosophers, did you notice any one philosopher periodically not consuming as much as the others? Try commenting out the printing of the state changing, and just look at the final totals.

1.3 Examples

Below are a few examples of the output. For brevity I have trimmed the output in the middle (note the "*A bunch more output*"). Your program will display all of the output for all of the philosophers, and it must be formatted as you see it below.

```
shell$ ./diners-v1 2
Time To Live (seconds)       :   2
Number of Dining Philosophers:   5
------------------------------
Philosopher  0: Think!
Philosopher  2: Think!
Philosopher  2: ......  Eat!
Philosopher  3: Think!
Philosopher  2: ......  ....  Done!
Philosopher  4: Think!
Philosopher  1: Think!
Philosopher  2: Think!
Philosopher  4: ......  Eat!
Philosopher  2: ......  Eat!
Philosopher  4: ......  ....  Done!
Philosopher  4: Think!
Philosopher  2: ......  ....  Done!
Philosopher  1: ......  Eat!

// ... A bunch more output

Philosopher  1: ......  ....  Done!
Philosopher  0: ......  Eat!
Philosopher  1: Think!
Philosopher  2: ......  Eat!
Philosopher  2: ......  ....  Done!
Philosopher  2: Think!
Philosopher  0: ......  ....  Done!
Philosopher  0: Think!
Philosopher  4: ......  Eat!
Philosopher  4: ......  ....  Done!
Philosopher  3: ......  Eat!
Philosopher  4: Think!
------------------------------
Philosopher   0: Ate 154 / Thought 155
Philosopher   1: Ate 155 / Thought 156
Philosopher   2: Ate 150 / Thought 151
Philosopher   3: Ate 146 / Thought 146
Philosopher   4: Ate 150 / Thought 151
------------------------------
```

```
shell$ ./diners-v2 1 3
Time To Live (seconds)       :   1
Number of Dining Philosophers:   3
-------------------------------
Philosopher  0: Think!
Philosopher  1: Think!
Philosopher  2: Think!
Philosopher  0: ......  Eat!
Philosopher  0: ......  ....  Done!
Philosopher  1: ......  Eat!
Philosopher  0: Think!
Philosopher  1: ......  ....  Done!
Philosopher  0: ......  Eat!
Philosopher  1: Think!
Philosopher  0: ......  ....  Done!
Philosopher  0: Think!
Philosopher  2: ......  Eat!
Philosopher  2: ......  ....  Done!

// ... A bunch more output

Philosopher  2: ......  Eat!
Philosopher  0: Think!
Philosopher  2: ......  ....  Done!
Philosopher  2: Think!
Philosopher  1: ......  Eat!
Philosopher  1: ......  ....  Done!
Philosopher  0: ......  Eat!
Philosopher  1: Think!
Philosopher  0: ......  ....  Done!
Philosopher  2: ......  Eat!
Philosopher  0: Think!
Philosopher  2: ......  ....  Done!
Philosopher  2: Think!
Philosopher  1: ......  Eat!
Philosopher  1: ......  ....  Done!
Philosopher  0: ......  Eat!
Philosopher  0: ......  ....  Done!
Philosopher  2: ......  Eat!
Philosopher  2: ......  ....  Done!
-------------------------------
Philosopher   0: Ate 100 / Thought 100
Philosopher   1: Ate 100 / Thought 100
Philosopher   2: Ate  99 / Thought  99
-------------------------------
```