SFU SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 300: Operating Systems I

Assignment 2

Due July 19, 2018

POLICIES:

1. **Coverage**
   Chapters 1-9
2. **Grade**
   10 points, 100% counted into the final grade
3. **Individual or Group**
   Individual based, but group discussion is allowed and encouraged
4. **Academic Honesty**
   Violation of academic honesty may result in a penalty more severe than zero credit for an assignment, a test, and/or an exam.
5. **Submission**
   Electronic project package via CourSys
   Programming language is not restricted unless otherwise stated.
   Provide a detailed readme with specific steps to run each submitted code.
   Exercise the code on CSIL computers and make sure it is RUNNABLE;
   Otherwise, the code may be subject to a grade of zero.
   TAs are not obligated to debug the submitted code.
6. **Late Submission**
   2-point deduction for late submission within one week;
   5-point deduction for late submission over one week;
   Deduction ceases upon zero.

QUESTIONS:

If any questions should arise, email Amineh (adadseta@sfu.ca) for Question 1 and Ali (ali_salman@sfu.ca) for Question 2. Thanks.

1. **6 points**
   You are required to implement an interactive version of the Dining Philosophers problem using the Resource Hierarchy solution to avoid deadlock. The Resource Hierarchy solution will be presented shortly. Five philosophers are numbered from 0 through 4. Each philosopher is at the state of thinking at the beginning.

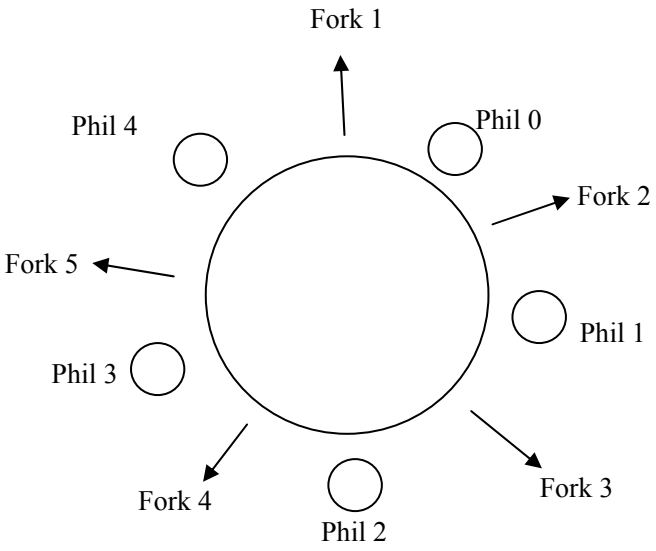
Resource hierarchy solution:

This solution to the problem is the one originally proposed by Dijkstra. It assigns a partial order to the resources (the forks, in this case), and establishes the convention that all resources will be requested in order, and that no two resources unrelated by order will ever be used by a single unit of work at the same time. Here, the resources (forks) will be numbered 1 through 5 and each unit of work (philosopher) will always pick up the lower-numbered fork first, and then the higher-numbered fork, from among the two forks they plan to use. The order in which each philosopher puts down the forks does not matter. In this case, if four of the five philosophers simultaneously pick up their lower-numbered fork, only the highest-numbered fork will remain on the table, so the fifth philosopher will not be able to pick up any fork. Moreover, only one philosopher will have access to that highest-numbered fork, so they will be able to eat using two forks.

Project Guidance and Requirements:

By being interactive, your program supports user inputs, which will determine which philosopher wants to eat or think.

Please follow the following figure to number philosophers and forks, that is, for Philosopher i, the left-side fork is Fork (i+2) mod 5, and the right-side fork is Fork i+1.



The commands for user interface should be interpreted as follows:

T 0 is a request to think from Philosopher 0;

E 1 is a request to eat from Philosopher 1.

In general, T i and Ei are a think request and an eat request from Philosopher i, respectively.

If the input is P, the output should be a representation of states of philosophers in turn and separated by one space, immediately followed by a '\n' at the end. 0

represents the state of thinking and 1 represents the state of eating.
For instance, entering P in the beginning of program execution would output 0 0 0 0 0\n. Entering ! should immediately terminate the program.

You are supposed to use Pthread library in C for implementing threads. Each philosopher is a thread that is waiting in the eating or thinking state before a user input targets them. Their change of the states is reflected in the entries of an array that is called states. You should also have a separate thread that reads the input and thus activate each philosopher thread based on the input. Make sure you won't confuse the case that a philosopher is waiting for an input and the case that a philosopher is waiting for a fork.
In order to make each philosopher blocking in a line of code in the eating or thinking condition, one can use pthread_cond_wait(pthread_cond_t &restrict cond, pthread_mutex_t *restrict mutex) function that is implemented in thread library. Also, for getting them off waiting, the main-thread can free that thread by using pthread_cond_signal(pthread_cond_t *cond).

Notes: Don't use printf and scanf functions in the thread functions. Use write(1, const void &buf, size_t count) and read(0, void *buf, size_tnbyte) instead.

Input and Output:
The input is always valid. In the case the user enters E i until Philosopher i managed to start eating or any other request from the first philosopher is omitted. The same goes with command T i. Also, if a philosopher is in the state of eating, a request for eating from that philosopher is omitted and vice versa.
The following is an example correct output for the given input:

```
adadseta@csil-cpu1:~/sfuhome/TA/summer 2018$ ./din
E 1
E 0
P
0 1 0 0 0
!
adadseta@csil-cpu1:~/sfuhome/TA/summer 2018$ ./din > output
```

Submission:

Put all your code in a single file named dinphil.c. Output as well as input interpretation should match exactly with what is described earlier and also entering ! should terminate the program.
If any of the above requirements are not satisfied or the program has additional printouts, there will be a deduction of 3 marks of correctness.

Guide:

It is required that you compile and run your program on CSIL computers whether onsite or remote. For using Pthread library, you might need to add the -lpthread at the end of the compilation command like gcc -o dinphil dinphil.c -lpthread.

[Grading Rubric: 1 point for successful compilation. 2 points for providing an example deadlock scenario that specifies the user inputs and the expected outputs for TAs to verify. 3 points for code correctness. REMEMBER to contact Amineh as soon as possible upon any questions, thanks.]

2. **4 points**

You are required to find the determinant of the given NxN matrix, where N>2, using multithreading.

The determinant of a matrix can be determined by calculating the determinant of its submatrices until N<=2. Divide the task of calculating the determinant of submatrices by creating multiple threads and joining the results at the end to compute the complete determinant.

Your input will be an integer N, where N>2. Create an NxN matrix using random numbers. Then calculate the determinant of the matrix using multithreading.

Some samples for you to verify program correctness.
a. Input(N): 4
possible random matrix generated by the program:
Matrix = [0 4 0 -3
1 1 5 2
1 -2 0 6
3 0 0 1]
Output:
Determinant: -250

b. Input(N): 4
possible random matrix generated by the program:
Matrix = [1 0 2 -1
3 0 0 5
2 1 4 -3
1 0 5 0]
Output:
Determinant: 30

[Grading Rubric: 1 point for successful compilation. 1 points for generating the correct outputs of determinant for the two samples provided above. 2 points for code correctness. REMEMBER to contact Ali as soon as possible upon any questions, thanks.]