Project 2: Operating Systems

Dining Philosophers

Due Date: as posted in Blackboard

Objective: This assignment will help you to understand thread programming and semaphores.

Part 1: The Classic Dining Philosophers Problem:

Five philosophers are seated around a circular table. Each philosopher has a plate in front of him. In the middle of the table is a bowl of spaghetti. The spaghetti is so slippery that a philosopher needs two forks to eat it. Between each pair of plates is one fork. The life of a philosopher consists of alternate periods of eating and thinking. When a philosopher gets hungry, he tries to pick up his left fork and his right fork, one at a time, in any order. No two philosophers may use the same fork at the same time. If the philosopher successfully acquires two forks, he proceeds to eat for a while. When he is done eating, he puts down his two forks and continues thinking. The only possible activities are eating and thinking.

The problem: devise a solution (algorithm) that will allow the philosophers to eat. The algorithm must satisfy mutual exclusion while avoiding deadlock and starvation.

Part 1 Specifications:

Your program should expand upon the code presented in the textbook for the Dining Philosophers problem. Your program should be able to handle n philosophers (where $1 < n \leq 15$), not just 5 philosophers. Your program should prompt the user for the number of philosophers to create. Your program should be implemented using threads (where you spawn off one thread for each philosopher), and thread semaphores. Your program should be deadlock and starvation free.

Program Output:

Your program should generate the following output to standard output (i.e., the display). When a philosopher begins an activity, you should print a message of the form:

```
Eating Activity
          1
01234567890
*  * *  * *     3 starts eating
```

In this example, there are 11 philosophers, and all the philosophers are thinking, except for philosphers 0, 3, 5, 8, and 10, which are eating. Note, you should never have the case where all philosophers are eating simultaneously. There are other conditions which should never occur as well (think about these conditions, as they will help you formulate the problem).

You should create a global *activity* array with one element per philosopher. A value of 0 means that the philosopher is engaged in thinking, and a value of 1 means that the philosopher is engaged in eating. You should create a semaphore *access_activity* which governs the reading and writing of data into this array. Each time a philosopher begins an activity, he should acquire this

semaphore, update the philosopher's element of the activity array, and then based on the values in the array, print out both activity displays in the format above.

As a longer example, consider the output lines:

```
Eating Activity
          1
01234567890
 *                1 starts eating
 *   *            4 starts eating
 *   * *          6 starts eating
     * *          1 ends eating
*    * *          0 starts eating
* * * *           2 starts eating
* *    *          4 ends eating
* *    *   *      9 starts eating
* *        *      6 ends eating
* *   *    *      5 starts eating
*     *    *      2 ends eating
      *    *      0 ends eating
      *           9 ends eating
    * *           3 starts eating
    * *     *    10 starts eating
    * *   * *     8 starts eating
    *     * *     5 ends eating
          * *     3 ends eating
          *      10 ends eating
 *        *       1 starts eating
 *                8 ends eating
 *      *         7 starts eating
 *   *  *         4 starts eating
```

Each line of output shows a start/stop in activity for one philosopher.

You should use a random number generator to determine the amount of time that a philosopher should be engaged in thinking, as well as how much time he should be engaged in eating (from the point in time he enters the eating critical section). The minimum activity period for thinking or eating should be 2 seconds. The maximum period for eating should be 5 seconds, and the maximum period for thinking should be 10 seconds (not including the time it takes to acquire the semaphores in order to enter the critical section).

In addition, your program should run until every philosopher has had a chance to think and eat at least 3 times. After this condition is satisfied, your program can stop and terminate. Be sure to clean up any remaining threads before your process/program exits.

Notes:

- To compile your program you should use something like:

  g++ prog3.cpp –lpthread

- Your program should not place unnecessary limits on the philosophers.
- Your program should ensure that no deadlocks occur and that no starvation occurs either.
- Programming style matters. I will take off points if your program exhibits poor programming style.
- Project submissions should be the same as before.