CS 74200 Project the Dining Philosophers Problem

This project implements a solution to the Dining Philosophers Problem using Pthreads **mutex locks, POSIX Semaphores**, and **POSIX condition variables** in the Chapter 7 Section 7.3.

The Dining-Philosophers Problem

In Section 7.1.3, we provide an outline of a solution to the dining-philosophers problem using monitors. This project involves implementing a solution to this problem using POSIX mutex locks and condition variables. Solutions will be based on thealgorithm illustrated in Figure 7.7.

```
/* Figure 7.7 Monitor Solution to Dinning Philosopher Problem */
monitor DiningPhilosophers
{
    enum { THINKING; HUNGRY, EATING) state [5] ;
    condition self [5];
    void pickup (int i) {
           state[i] = HUNGRY;
           test(i);
           if (state[i] != EATING) self[i].wait;
    }

    void putdown (int i) {
           state[i] = THINKING;
          // test left and right neighbors
            test((i + 4) % 5);
            test((i + 1) % 5);
    }

    void test (int i) {
        if ((state[(i + 4) % 5] != EATING) &&
            (state[i] == HUNGRY) &&
            (state[(i + 1) % 5] != EATING) ) {
          state[i] = EATING ;
          self[i].signal () ;
        }
    }

    initialization_code() {
              for (int i = 0; i < 5; i++)
                  state[i] = THINKING;
        }
    }
```

/* Figure 7.6 Structure of Philosopher / */

```
   while (true){
      pickup (i)

         /* eat for awhile */

      putdown (i)

         /* think for awhile */
   }
```

Th implementation requires creating five philosophers, each identified by a number 0 . . 4. Each philosopher will run as a separate thread. Philosophers alternate between thinking and eating. To simulate both activities, have each thread sleep for a random real number period between 1 and 4 seconds.

Thread creation using Pthreads is covered in Section 4.4.1. When a philosopher wishes to eat, he or she invokes the function

```
pickup_forks(int philosopher_number)
```

where `philosopher_number` identifies the number of the philosopher wishing to eat.

When a philosopher finishes eating, he or she invokes

```
return_forks(int philosopher_number)
```

Your implementation will require the use of POSIX condition variables, which are covered in Section 7.3.

Project Description

The main function will create PHILOSOPHER_NUM (5) threads for philosophers and will sleep for a period of time (run_time), and upon awakening, will terminate all threads and print out the numbers of meals each philosopher eats and calculate the minimum, the maximum, and the average numbers of meals eaten among philosophers. A philosopher thread exits when the MAX_MEALS (10) meals have been completed. Use sleep function for random one to MAX_THINK_EAT_SEC (4) **double value** seconds thinking and eating time. The main function will be passed one parameter run_time on the command line. The program may be named dinning_philos.c.

To simulate the monitor solution, a mutex lock must be used to lock the state and condition variables at the start and unlock at the end in pickup_forks and return_forks functions. The condition can be state[i] == EATING for ith philosopher. Please refer Section 7.3 for how mutex locks and condition variables are used in POSIX.

Program Environment

- Install Linux virtual Machine with VirtualBox provided by the textbook
  http://cs.westminstercollege.edu/~greg/osc10e/vm/index.html.
  Install Linux desktop >sudo apt-get install ubuntu-desktop if needed.
- Native Linux environment
- Window Subsystem for Linux with pthreads support

Reading

- Read 6.7.2 about implementing a Monitor using semaphores.

Report

A report should include the following table

Run Time (sec)	Minimum Meals	Maximum Meals	Average Meals
30			
40			
50			
60			
70			
80			

Submission

1. Provide the report, the source code file and the output file (using Linux script command) in a zipped file
2. Submit the zipped file as an attachment.

Grading

Grading Rubric	
Project	100 points Maximum 30 points if the program does not compile and run.