CSS 430 Operating Systems
Program 3A: Dining Philosophers Problem

1. Purpose

This assignment solves the dining philosophers problem, using the pthread library. You will exercise how to use: pthread_mutex_init/lock/unlock and pthread_cond_init/wait/signal to implement a monitor.

2. Synchronization among Dining Philosophers

Review the textbook (9th edition) section 5.7.3 and section 5.8.2. As explained in the textbook, the semaphore solution to associate each semaphore or lock to a different chopstick creates a deadlock. The ***best solution*** is to use a monitor that guarantees each philosopher to pick up both left and right chopsticks simultaneously. An ***alternative but quite inefficient solution*** is to let a philosopher hog the entire table so that any other philosopher must wait until s/he is done with eating. While it's easy, it does not allow any concurrency among the philosophers.

In this assignment, you will implement these two solutions with pthreads and empirically compare the performance of their concurrent executions. You can find *philosophers.cpp* from Canvas File folder: Files/code/prog3/philosophers.cpp. This program includes:

Classes, Threads, and Main Program	Actions
class Table0	Allows each philosopher to pick up and put down his/her left/right chopsticks without checking their availability. The logic is wrong but runs the fastest.
class Table1	Allows each philosopher to lock the entire table before picking up his/her chopsticks and unlock the table after putting down the chopsticks. The logic is correct but has no concurrency, (i.e., runs the slowest). You need to complete this table lock/unlock.
class Table2	Allows each philosopher to pick up and put down his/her left/right chopsticks through a monitor solution. You need to implement the logic using pthread_mutex_init/lock/unlock and pthread_cond_init/wait/signal. Please refer to textbook section 5.8.2 to complete your code.
void *philosopher(void *arg)	Run as an independent pthread and performs three eating cycles at a specific table, 0, 1, or 2. Note MEALS = 3.
int main(int argc, char** argv)	Receives 0, 1, or 2 as the 1st argument, which means the table #; launches 5 philosopher threads as passing the table #; waits for their completions; and prints out the elapsed time. Note PHILOSOPHERS = 5.

3. Statement of Work

Task 1: Complete the Table1 class by inserting pthread_mutex_lock/unlock appropriately.
Task 2: Complete the Table2 class using pthread_mutex_init/lock/unlock and pthread_cond_init/wait/signal.
Task 3: Compile and run your a.out as follows:

```
$ g++ -lpthread philosopher.cpp
$ ./a.out 0
...
time = 3001444
$ ./a.out 1
...
time = 15005909
$ ./a.out 2
...
time = 8003182
$
```

4. What to Turn in

Total 20pts.

	Materials	Points	Note
1	philosopher.cpp a. Code organization: **+2pts** • Well organized: 2pts, • Poor comments or bad organization: 1pt, or • No comments and horrible code: 0pts b. Correctness: **+13pts** • Table1 implemented the table lock and unlock: **+5pt and** • Table2 implemented the textbook figure 5.18 correctly: **+8pts,** o Correct: 8pts, o Incorrect: 4pts, or o No implementation: 0pts	15	Submit this .cpp file individually
2	An execution snapshot • Correct (1 thread at a time with table #1 and 2 threads at a time with table #2): 5pts • Wrong: 3pts, or • No illustrations: 0pts	5	Include the snapshots in a pdf file named: FirstNameLastName_prog3A.pdf. Submit this pdf file individually. Please clearly label each snapshot and add necessary explanation.