Operating Systems - II: CS3523

Programming Assignment - V

Implement Dining Philosophers using Conditional Variables

Assignment Report

Sagar Jain - CS17BTECH11034

April 9, 2019

Contents

Program Design

In this assignment we solve the dining philopher's problem using a monitor with ***conditional variables***implementation. The program design is explained in the following points, we follow the function main in the explaination:

- We begin by opening the required I/O files.
- We then read input from the input file. After reading the input we call the initialization function in the monitor which requires the number of philosophers.
- In the next few lines we have the setup for the pthreads i.e. declaring attributes, setting scope.
- We have two vectors ***philosopher_threads*** and ***philosopher_thread_ids*** for the pthreads themselves and the ids of the threads respectively.
- Using a loop we then launch all the pthreads, following which is another loop which joins all the pthreads when they are done.
- The philosopher thread is described in the below points.
  - First we type cast the **void* param** to an **int*** to give us the id of the pthreads we are in.
  - Then we declare the variables in which we store the time values following which we define the random number generators and the exponential distributions for the eating time and the thinking time.
  - In the loop we use the monitor functions(***pickup*** and ***putdown***) to enter and exit the critical sections. We use ***fprintf*** to log into files since it does not have concurrency issues. We use ***usleep*** to sleep the threads for eating and thinking.
  - We use choronos for all time related variables. We also use the funtion ***get_formatted_time*** to get the string which we log as time into the log file.
- The monitor ***DiningPhilosophers*** and ***conditional variables*** are described in the following points.
  - The monitor has a state for variable for each of the philosophers, which denotes if they are thinking, hungry or eating.

- We use ***pthread_mutex_t*** and ***pthread_cond_t*** to ensure mutual exlclusion within the monitor, the conditional variable ***self*** is assosciated to the mutex ***mutex***.
- We have a ***vector***$< pthread\_cond\_t >$ called ***self*** which has one conditional variable for every philosopher.
- The function ***pickup*** is used to enter the critical section by the philosopher threads, in which they use the function ***test*** to verify if both the chopsticks are available else it waits.
- The function ***putdown*** is used to exit the critical section it also signals the waiting philosophers on either side of the current philosopher using the ***test*** function with different arguments.
- The ***DiningPhilosophersinit*** function is used to initialize the states of the philosophers and to resize the vectors with the correct number of philosophers.

Program Output

Each of the programs output two files. One file is the log file containing discrete events of the threads entering and exiting the critical section and another file is the file which contains the statistics related to the waiting times of the threads.

Example Output:

Log File:

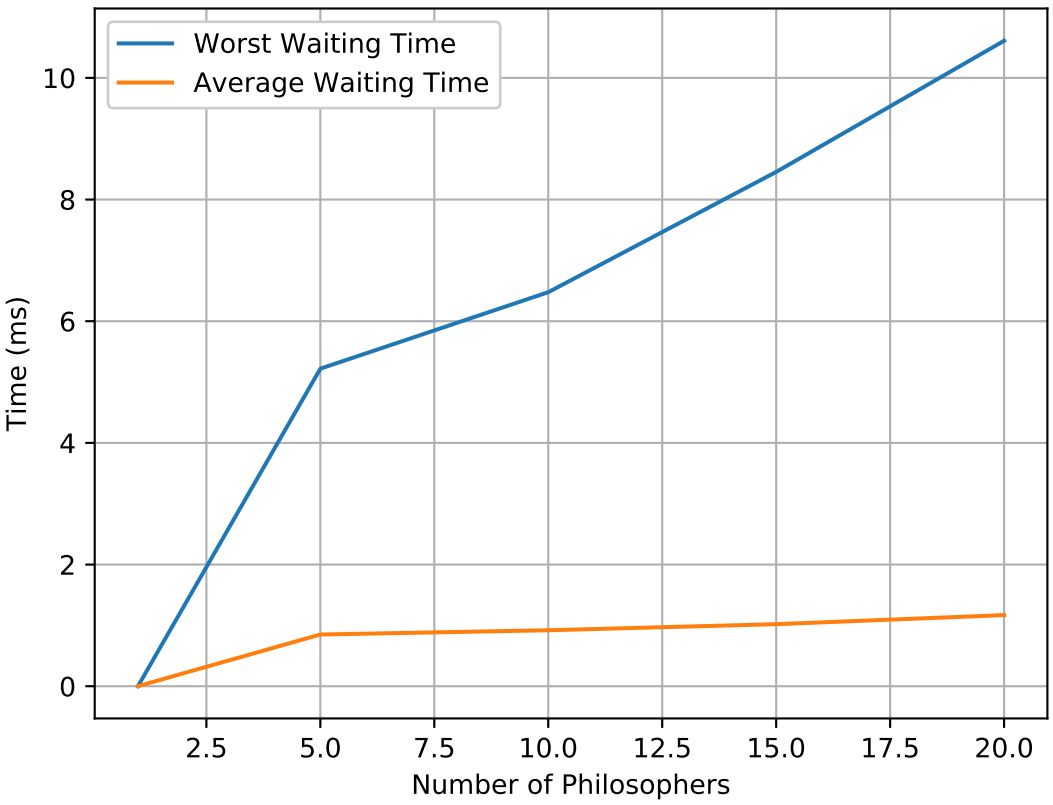
3 th CS entry by Philosopher thread 2 at 23:08:21.
1 th eat request by Philosopher thread 5 at 23:08:21.
3 th CS exit by Philosopher thread 2 at 23:08:21.
4 th eat request by Philosopher thread 2 at 23:08:21.
4 th CS entry by Philosopher thread 2 at 23:08:21.
4 th CS exit by Philosopher thread 2 at 23:08:21.

Average File:

Average Time Taken for Philosopher Threads 2.88679 ms
Worst Waiting Time for Philosopher Threads 34.132 ms

Results & Graphs

Average & Worst Waiting Times



Average Waiting Time goes from 0.00105 to 0.25

Worst Waiting Time goes from 0.002 to 12.451

Explaination of Results

The results are not suprising and for the most part are in the range of our expectations, the folllowing points are worth noting:

- The average waiting times are much lesser that the worst waiting times consistently. This implies that in our solution there is a possibility of starvation.($WorstWaitingTime/AverageWaitingTime > 100$ consistently!)
- The Average Waiting Time increases very slowly, this could imply that average waiting time is not as dependent on the number of philosophers as worst waiting time. This can be explained by the fact that at any moment we have around n/2 philosophers who are eating irrespective of the value of n.
  - The average time for dining philosophers should not be dependent on the number of philosophers, the slight increase that we do see in the times is due to the fact that all need to aquire the same mutex lock.
- The erratic behavior of the worst case waiting time is due to the fact that there can always be one thread which keeps missing out on the chance to eat even for small value of n, in the graph presented we have a relatively linear graph but it may not always be so and that is within the bounds of our expectations.
- The worst case wait times increase because more number of threads are waiting on the same mutex lock and we have a ***convoy effect*** on the waiting times of all the threads which are waiting on the mutex lock.
- This implementation can be made free of starvation by using an assymetric way to pick up chopsticks and removing the policy of picking up chopsticks only when both the philosophers beside the philosopher are not eating.