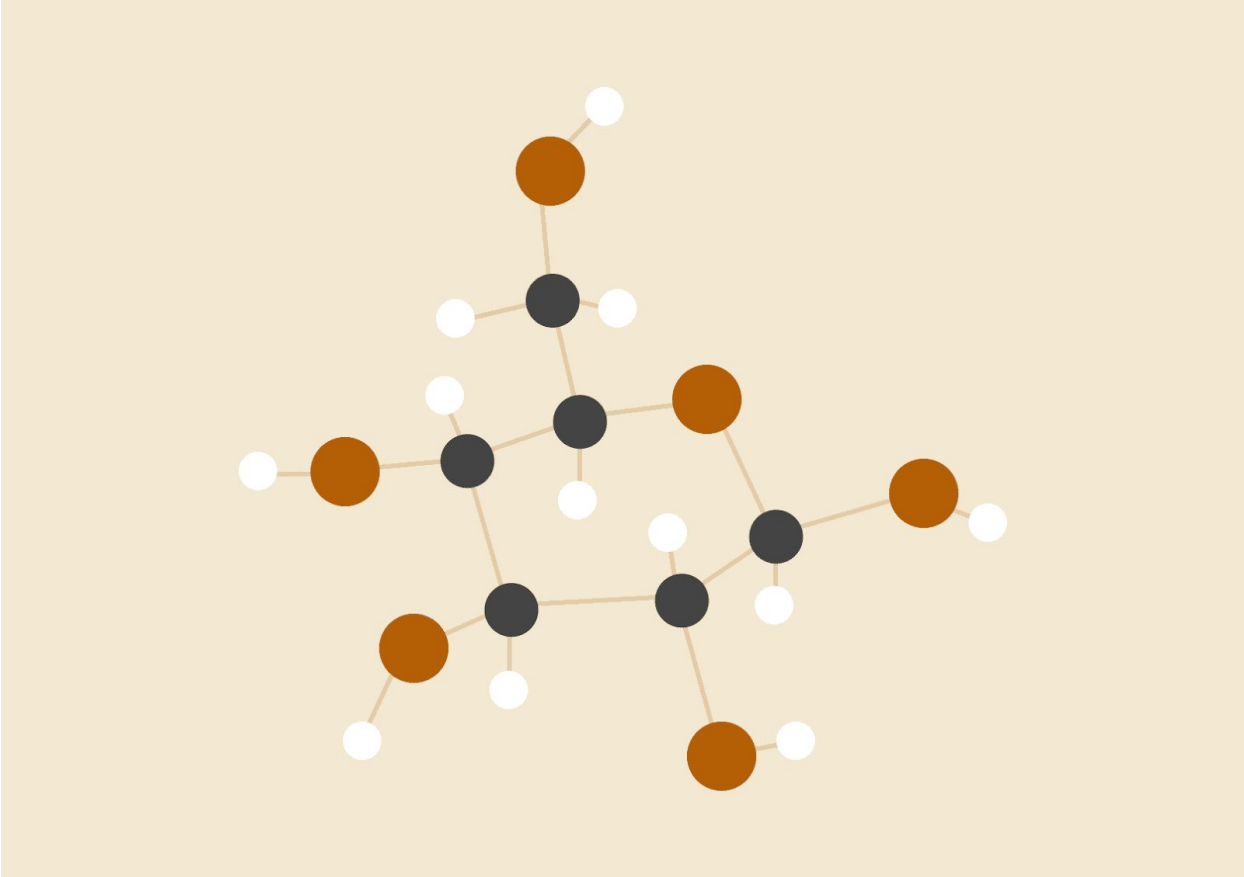
OS ASSIGNMENT 5

Solution of dining philosopher using conditional variables



Happy mahto

09.04.2019

CS17BTECH11018

INTRODUCTION

The program solves dining philosophers problem using conditional variables.

HYPOTHESIS

Critical Section: Segment of code which is supposed to be accessed by only one thread at a time.

Entry Section: Segment of code which is available to all the threads and where threads contend to enter in the critical section. Bounded waits guarantee that no threads wait for forever in the critical section and gets a fair chance to enter in the critical section.

Exit Section: Segment of code wherein thread exits critical and opens the lock for other processes to enter in the critical section.

Waiting Time: the time that threads spend in entry section waiting to enter in the critical section.

Conditional variable: **Condition variables** are synchronization primitives that enable threads to wait until a particular **condition** occurs. **Condition variables** are user-mode objects that cannot be shared across processes. **Condition variables** enable threads to atomically release a lock and enter the sleeping state.

IMPLEMENTATION

In the solution of dining philosopher solutions, philosophers with even no try to pick up the fork with first right and then left, and philosopher with odd no tries to pick up the fork with first left then right.

- Programs use input file “input.txt” to accept input values. It accepts no of philosophers, no of time philosophers want to eat, time taken to eat and time taken to think.
- Program uses default_random_generator generator to generate a random number with exponential_distribution, they are contained in distribution1 and distribution2.

- Main threads create “no_of_philosophers” no of threads. Each philosopher's threads execute the Philosophers funtion “no_of_eatings” times.
- Conditional variable is implemented using pthread library and it uses a mutex to perform modification while the lock is held.It is used when threads need to wait for a resource to become available. A thread can "wait" on a CV and then the resource producer can "signal" the variable, in which case the threads who wait for the CV get notified and can continue execution. A mutex is combined with CV to avoid the race condition where a thread starts to wait on a CV at the same time another thread wants to signal it; then it is not controllable whether the signal is delivered or gets lost.
- Inialization of conditional variables

```
pthread_cond_init(&self[i], NULL);
```

- Conditional wait

```
pthread_mutex_lock(&lock);
if(state[philosopher] != EATING){
        pthread_cond_wait(&self[philosopher], &lock);
}
pthread_mutex_unlock(&lock);
```

- Conditional signal

```
pthread_mutex_lock(&lock);
        if(state[philosopher] != EATING){
                pthread_cond_signal(&self[philosopher]);
        }
        pthread_mutex_unlock(&lock);
```

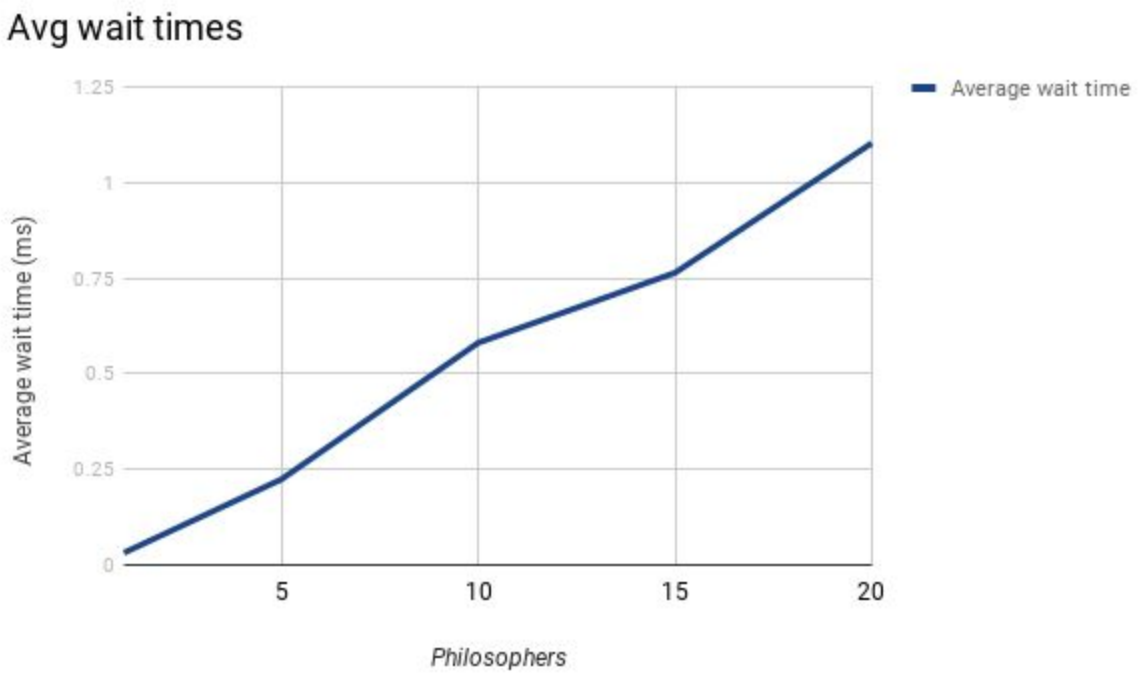
- pickup() function lets the thread to access forks if no other philosopher already has accessed it, it checks it using the test function.
- test() function checks whether the adjacent philosopher threads already have the fork or not if then it allocates the forks and else makes it wait and set the state of

philosophers to hungry.

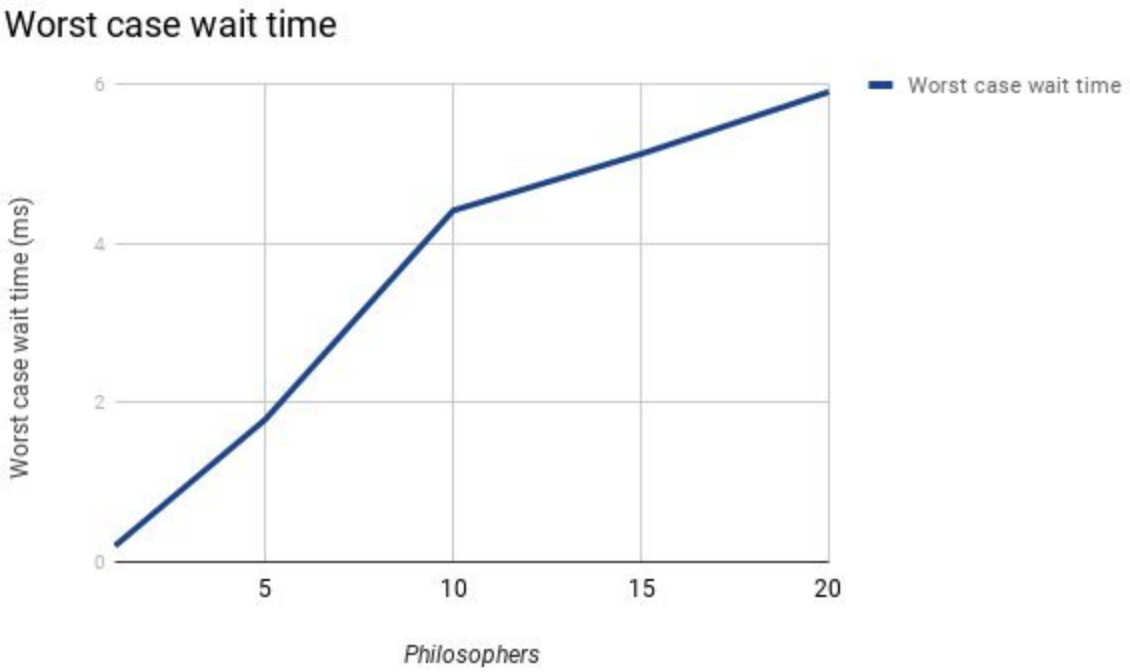
- putdown() function is used to release the forks accessed by the philosophers.
- Worst time is calculated using clock() function present in systime library.
- Enum is used to enumerates the thread state to either Hungry, Eating, or Thinking.
- Average wait time is the average waiting time a philosopher while requesting for the forks.

GRAPH

- Graph between the average waiting time of philosopher and no of philosopher



- Graph between worst waiting time for philosopher and no of philosophers



RESULTS

Graphs show us the the average waiting increase with an increase in no of philosophers, and same happens for the worst waiting time for philosopher to access the forks.

Worst case waiting time is nearly 5 times the average waiting time suggesting that the threads may starve, due to alternate request from the philosophers while the middle is starving.

Deadlock is avoided using the methods that odd numbered philosopher picks his left fork first and then right fork, wheareas a even numbered philosopher picks his right fork first and then left fork.

From the graphs we can see that there is no deadlock, while running the test.

REFERENCES

- Operating systems by ABRAHAM SILBERSCHATZ, PETER BAER GALVIN, GREG

GAGNE.

- Libraries used
    1. <pthread.h> : used for creation of thread and work assignments
    2. <atomic> : Used for making the lock atomic so that only one thread can edit it in a single time
    3. <random> : Used for the random number generator for seeding the sleep interval.