Operating Systems–2

CS 3510 Spring 2019

Programming Assignment 5:

<u>Implement Dining Philosopher's using Conditional</u>

<u>Statements</u>

INSTRUCTOR: DR. SATHYA PERI

Report By:
Vijay Tadikamalla
CS17BTECH11040

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Task

To solve the Dining Philosopher's problem using conditional variables as discussed in
the class in C++. You have to implement these two algorithms and compare the average and worst-case times taken for each thread to access the critical section (shared resources).

Approach and Implementation

1. To achieve our above-mentioned goal make two functions producer and consumer which takes thread_index as a parameter. We will pass this function along with the thread index to each thread and calculate the average time for both semaphore and mutex.
2. Functions and data-types of the chrono library (and other libraries) like
    a. **std::chrono::system_clock, std::chrono::system_clock::now()**
    b. **std::chrono::time_point**
    c. **struct tm - Time structure**
    d. **struct tm *localtime(const time_t *timer)**

    were used to calculate the average waiting time and max waiting time.
3. **int usleep(useconds_t *usec*)** function was used to suspend the execution of the thread for microsecond intervals
4. **template <class RealType = double> class exponential_distribution**: This is a random number distribution that produces floating-point values according to an exponential distribution, which is described by the following probability density function:

$$p(x|\lambda) = \lambda e^{-\lambda x} \quad , \; x > 0$$

5. We make two exponential distributions and pass the value of 1/µp, 1/µc in the constructor. Later this can be used to obtain random numbers t1 and t2 with values that are exponentially distributed with an average of µp, µc seconds.

```
distribution1 = new
exponential_distribution<double>(1/mu_p);
distribution2 = new
exponential_distribution<double>(1/mu_c);
```

6. In Dining philosopher problem we use the conditional variable and locks to ensure that two philosophers pick up chopsticks at the same time.
7. At the same time, we allow multiple philosophers to eat at the same time provided they at not neighbors
8. One lock “mutex” is used in this implementation.
9. One array of size equal to no of philosophers is used to represent the three states of philosophers.
10. N conditional variables are used in this implementation. Each conditional variable corresponds to one philosopher.
11. This implementation of Dining philosopher problem prevents deadlocks but it does not prevent starvation.

Output analysis of Graph

- Both Average time taken and the worst case time by philosophers increase as the no of philosophers increase
- Average time taken by philosophers almost remains constant for high philosopher count.