**Week 8 assignment**

Couse: Operating Systems Design(146043-31001), 2021

Name: Chang wan Woo

Programming Assignment NUMBER: Implementation assignment #3

Due DATE: 2021/05/11

**Program environment**

Window 10, Interllij, java version "1.8.0\_291",

Java(TM) SE Runtime Environment (build 1.8.0\_291-b10)

**Problem Statement** :

In dining philosopher problems, deadlock and starvation are the main problems to prevent. I implement preventing deadlock by checking left and right chopsticks whether it is available, and also, I prevent starvation by checking “STARVATION\_COUNT” which is predefined in miscsubs.java.

**Software design:**

In my project, there are four classes, miscsubs.java, dining, java, Philosopher.java, and Chopstick.java. I constructed Philosopher and Chopstick, former class implement Runnable interface to make Philosophers as Thread. Philosophers own both chopsticks only both chopsticks are available. By doing this, program prevent deadlock because chopsticks are only assigned to philosophers when both are available. Also, the “synchronized” which is supported by JAVA prevent deadlock by preventing more than two philosophers access canChopsticks() and putdown() method. In addition, when they take chopsticks, they check whether there are philosopher whose starvation count is bigger than 15.

One more thing, I made my program to print out when TotalEats>MAX\_EATS which implies that program is terminated. So we can see results after philosophers taking 500 counts.

**How to start?**

I attached dining.zip which stores java files which is needed to execute my program, you can start by unzip my file and run my main class. You can see results in command tab.

**Implementation**

**class Description:**

**Class dining**

This is the main class which supports other class and make philosophers threads. After program is finished, Philosopher class informs to main class by calling notify() so that it prints out how many philosophers eat.

**Class Philosopher**

In philosopher class, I used while loop to let philosophers eat until miscsubs.TotalEats <miscsubs.MAX\_EATS. I consider two things: deadlock and starvation. I prevented deadlock by synchronized(object) and they can hold chopsticks only both are available. Also, in worst case, one philosophers can starve while program execution. So I checked startvation\_count for every while loop.

**Class Chopstick**

There are also 5 chopsticks for philosophers. I defined lock Object to synchronize chopsticks.

Method: canChosptic(int id)

Input: int id

Output: void

If one of chopstick[leftChopstick] or chopstick[rightChopstick] is false, it is locked by calling lock.wait() and other thread notify them when they call putDown() method.

Method putDown

Input: int id

Output: void

The putDown() method unlock synchronization by calling notify(). Philosophers putDown both left chopstick and right chopstick.

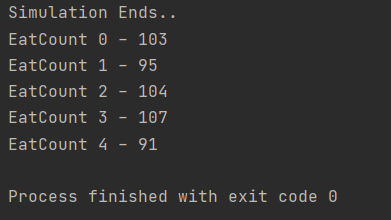
Class: miscsubs

It is predefined several method which is necessary for implementing philosophers. I don’t edit method in this class.

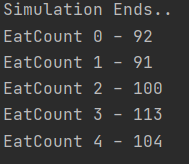
**Test Description and Results**

**In this section we can see 5 philosophers took about 100 respectively. There are variances but all philosophers took range from 80~120.**

**Test1:**



**Test 2**



**Test 3**

