Benjamin Le Heup

Professor Damian Dechev

January 25, 2020

Homework Assignment 1

Problem 1)

In findAllPrimes.cpp, I decided to split all the work between the 8 thread evenly by having each thread start at an odd number starting from 3 to 17. So thread 1 will be checking if 2 is a prime number, thread 2 will check if 5 is a prime number and this is done up to 17. After each thread checks if the current number is prime or not it will add it to a vector if it is a prime number this vector list starts with the value 2 in the list since we know that two is a prime number. Once it is finished it increments the number it is checking by 16 which means after the first thread is finish checking 3 it will check 19. By doing this it prevents the possibility of any number being check twice as well as being able to check every odd number up until 100,000,000. When a number is prime then it is added to a total variable that will keep track of the sum of all the primes. The method I used to check if a number is prime is by checking all odd numbers to see if the number is a multiple of any of the numbers up to the sqrt of that number. I do not check 2 since all the numbers being check are odd numbers.

Problem 2)

Version 1)

In this version of the program, I made the philosophers run concurrently with each other and have them think and eat if they are hungry. This is not safe from deadlock since there is a possibility that all the threads or philosophers all pick up the left chopstick before any of the philosophers pick up a right chopstick. This case is still unlikely since I have thinking last 500ms and eating last 100ms so it would require just perfect timing with executions of certain statements taking longer or shorter than others.

Version 2)

In this version of the programming, it is virtually the same as the first version except that I allocate the ordering of picking up chopsticks differently. I had the odd number philosophers pick up the right chopstick first and the even number philosophers pick up the left chopstick first. This prevents deadlock since there is not the possibility that all philosopher holing a chopstick that the other one needs. If philosopher 1 tries to get chopstick 1 and then to chopstick 2 the only way chopstick 2 would not be available if philosopher 2 needed chopstick 2 and therefore would have started eating making chopstick 2 available for philosopher 1 to eat after he is done. This holds for all philosophers and eliminates any chance of deadlock.

Version3)

In this version, I implement a queue to guarantee that one person eats at a time. This made it so that whenever a person gets hungry, they are instantly put into a queue and start eating when it is in front of the line. I used one lock on both the push and pop functions to prevent multiple threads from accessing the same queue at the same time.

Version4)

In this version, it is exactly like version 3 but takes in input from the command line and creates Philosophers and Chopsticks arrays accordingly.