James Hund
TJ Moore

Project Report

Algorithm 6.10

Functions:

Fork: This function creates an instance of a fork. A fork can be taken and released. When a fork is taken that fork is blocked and no other philosopher can have possession at the same time. When a fork is released then it can be taken by a different philosopher.

Philosopher Class: This class creates a thread for each philosopher. Inside this class it creates instances of a philosopher and their actions.

Philosopher: This describes each philosopher as having a name, left fork, right fork, and a round number.

Think: This function defines the action think. It puts the philosopher to sleep for a random amount of time. This is called when the philosopher is done eating and after the philosopher has released control of both forks.

Run: This starts each action of each philosopher.

Eat: Checks if the left fork is in use. If it is then the philosopher must wait. If not in use then it will take the left fork then the right. It will eat for a random amount of time afterward it will release both forks.

This algorithm keeps all the constraints.

This algorithm has the potential to suffer from deadlock if all of the philosophers take the left fork immediately when they come in. This will make them wait indefinitely. It can also suffer from starvation if the same philosopher keeps eating, releasing, and grabbing the forks over and over. This is solved with the number of rounds argument.

It does maintain mutual exclusion and efficient behavior in the absence of contention.

James Hund
TJ Moore

Project Report

Problems Encountered: Getting await and signal to work nicely. Kept getting deadlocks while trying to implement them.

Sources:https://github.com/rupakraj/dining-philosopher/blob/master/Dine.java
https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Semaphore.html

Algorithm 6.12

Functions:

Fork: This function creates an instance of a fork. A fork can be taken and released. When a fork is taken that fork is blocked and no other philosopher can have possession at the same time. When a fork is released then it can be taken by a philosopher.

Philosopher Class: This class creates a thread for each philosopher. Inside this class it creates instances of a philosopher and their actions.

Philosopher: This describes each philosopher as having a name, left fork, right fork, and a round number.

Think: This function defines the action think. It puts the philosopher to sleep for a random amount of time. This is called when the philosopher is done eating and after the philosopher has released control of both forks.

Run: This starts each action of each philosopher.

Eat: Checks if the left fork is in use. If it is then the philosopher must wait. If not in use then it will take the left fork then the right. It will eat for a random amount of time afterward it will release both forks.

This algorithm keeps all the constraints.

James Hund
TJ Moore

Project Report

Starvation is the only thing that can happen if looped forever, but since the implementation of rounds in the code it is free of starvation.

It does maintain mutual exclusion and efficient behavior in the absence of contention.

Problems Encountered: Getting await and signal to work nicely. Kept getting deadlocks while trying to implement them.

Sources: https://github.com/rupakraj/dining-philosopher/blob/master/Dine.java
https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Semaphore.html