UCT CSC3002F 2019

Operating Systems Assignment 2: Synchronization

Part III (of III)

(version 1)

Lecturer: Michelle Kuttel

Due dates:

Thurs 9 May 2019, 9am (Part III)

Aim

This assignment aims to give you experience in thread (and, by extension, process) synchronization using **semaphores**. In this project, you will write multithreaded programs in Java to solve **three classic synchronization problems**, using only semaphores. A secondary aim is to give you experience in reading, understanding and correcting existing code. The assignment is in three parts, increasing in difficulty. The third part is described below.

Part III: Making methane with semaphores

Methane is a molecule with four hydrogens attached to a single carbon (CH_4). For this project, you will run a simulation of the combination of carbon and hydrogen atoms into methane molecules.

There are two kinds of atoms (threads), *carbon* and *hydrogen*, in your simulations. In order to assemble these atoms into methane molecules, each atom must wait until the right number (and type) of atoms are available to bond (by calling the `bond()` method) into a complete methane molecule.

There are a number of synchronization constraints on this simulation, as follows.

- **Only one methane molecule forms at a time.** You must guarantee that all the atoms from one molecule invoke `bond()` before any of the atoms from the next molecule. In other words:
- If a carbon thread arrives at the barrier when no *hydrogen* threads are present, it has to wait for four *hydrogen* threads.
- If a *hydrogen* thread arrives at the barrier when no other threads are present, it has to wait for a *carbon* thread and a further three hydrogen threads. Etc.

You do not have to matching the atoms (threads) up explicitly; that is, the atoms (threads) do not necessarily know which other atoms (threads) they are paired up with. The key is just that atoms pass the barrier in complete sets; thus, if we examine the sequence of atoms that invoke `bond()` and divide them into groups of five, each group should contain one *carbon* and four *hydrogen* threads.

Assignment: Write synchronization code in Java for *carbon* and *hydrogen* molecules that enforces these constraints, using **only** the **semaphore construct**. We will provide a reusable barrier class (an extension of Part I) for the barrier you will need.

1.1 Code skeleton: molecule

You **must use** and extend the `molecule` **package provided**, correctly implementing the classes `Hydrogen` and `Oxygen` (and all the methods), so that the `RunSimulation` class will execute correctly, **always**. You may **not add any methods to these classes**. The semaphores you will need are defined already in the `Methane` class. **However, do not alter** this class or **any anther class at all** (although you must submit them). You will need to inspect the various classes to see how they work – this is part of the assignment and **no explanation other than the code will be given**.

An **example** of a correct execution of `molecule` is:

- java molecule.RunSimulation 12 3

```
Starting simulation with 12 H and 3 C
---Group ready for bonding---
...Bonding....H4
...Bonding....H3
...Bonding....H1
...Bonding....H2
...Bonding....C2
---Group ready for bonding---
...Bonding....H7
...Bonding....H6
...Bonding....H5
...Bonding....H10
...Bonding....C1
---Group ready for bonding---
...Bonding....H12
...Bonding....H11
...Bonding....C3
...Bonding....H9
...Bonding....H8
```

1 Code requirements

You will code your solutions in Java, adding to the skeleton code provided.

You may only use the **Semaphore** class in the `java.util.concurrent` library: **no other Java synchronization constructs at all**.

All code must be **deadlock free.**

The output must comply with the stated synchronization constraints and with the examples shown.

2 Assignment submission requirements and assessment rules

- You will need to create, **regularly update**, and submit a GIT archive of your code for each separate part (i.e. one archive for Part I, one for Part II and one for Part III).

- You are required to **extend the provided code templates**. If you fail to do this (i.e. submit alternative code), you will obtain a mark of 0 for the assignment.
- Each submission archive must include a **Makefile and README file** (including running/installation instructions) for compilation/running.
- Label your assignment archive with your student number and the assignment number e.g. KTTMIC004_CSC3002S_OS2_PartI
- Upload the files and **then check that they are uploaded.** It is your responsibility to check that the uploaded file is correct, as mistakes cannot be corrected after the due date.
- The usual late penalties of **10% a day (or part thereof)** apply to this assignment.

- The **deadline for marking queries** on your assignment is **one week after the return of your mark.** After this time, you may not query your mark.
- Note well: submitted code that does not run or does not pass standard test cases will result in a **mark of zero**.
- **Any plagiarism, academic dishonesty, or falsifying of results reported will get a mark of 0 and be submitted to the university court**.