PCPP - Assignment 1

Dallas Trent Maxwell (dalm@itu.dk)
Martin Conradsen (conr@itu.dk)
Mohammad Toheed Sharif (mosh@itu.dk)

August 30, 2021

Exercise 1

Mandatory

1. The output values are random (less than 20 million), and not the expected output of 20 million.

2. The value is 200. We believe that this is due to the small amount of loops, meaning that the first thread completes 100 iterations before the second thread begins. There is no guarantee that this will always happen - it depends on the scheduler and other factors such as the operating system and compiler. There are interleavings where this would not be the case.

3. There is no difference when using the different forms. This is because the statements are shorthand for each other, and they execute the same set of instructions, and are not atomic. That is, the value of `count` is read, copied to a temporary variable, and then incremented by one no matter which form is used. By running the code with the different forms of assignment, it made no difference to the counter.

4. By adding a lock in the critical section of the `increment()` method, we ensure mutual exclusion. By using locks, we avoid a data race when both threads attempt to read and/or write the `count` variable. This is the case in all interleavings, which is why no other output is possible.

Challenging

5. After decompiling the `.class` file with each of the three forms, it is clear that there are no differences between the decompilations, which verifies the explanation provided in exercise 1.3.

Exercise 2

Mandatory

1. Program is included in the source files (name: *TestPrint.java*).

2. Thread *t1* is started with `t1.start()`. Thread A prints a dash. Thread *t2* is started with `t2.start()`. Since *t1* is still running, it prints another dash before *t2* has the chance to print a vertical bar. There are now two dashes. *t2* then prints a vertical bar.

3. As explained in exercise 1.4, in this case we determine the critical section as the two print statements. By locking this section, we insure that `print("-")` happens before `print("|")`.

Exercise 3

Mandatory

1. Program is included in the source files (name: *CounterThreads2Covid.java*).

2. Same explanation as in exercise 1.4. By including the if-statement in the critical section, we ensure that the value is correct. Therefore we will never reach an amount over 15000, which can happen if the `if-statement` is not part of the critical section.

Exercise 3

Mandatory

1. We believe that the categories in both sources are too similar to find an example of a system that is included in one set of categories but not in the other.

Exploitation of multiprocessors in the notes is equal to *convenience* in Goetz, since the motivation for writing multiple programs that perform a single task (instead of one program that performs multiple tasks) is to exploit the fact that a computer has multiple processors. In the case of a single-processor system, programs that performed multiple tasks in succession would not be slower than multiple programs performing single tasks, since these would also be in succession.

Intrisic parallelism in the notes is equal to *resource utilization* in Goetz, since both categories concerns doing another task while waiting for one to finish (e.g. like we can do other tasks while waiting for the water to boil in the real world, so can the computer do other tasks while waiting for a task to finish).

Concealed parallelism in the notes is equal to *fairness* in Goetz, since both categories concern letting programs share the computer's resources to act as if each program runs to completion without interruptions or having to wait for another program to start and/or finish.

2.

Exploitation:

- A program to calculate all prime numbers under 1 million, dividing the ranges between different processors, allowing multiple calculations to happen concurrently.
- A program that shows a spinner while fetching a resource from the web (such as an image), and then removes the spinner once the fetching is completed.
- Letting the user use other elements of a GUI while waiting for a long task to complete (e.g. pressing the Cancel-button while a task is running).

Inherent/Intrinsic:

- An oven that plays a sound when it has fully heated to the chosen temperature (instead of turning off the heating to play the sound).
- A robot getting information from different sensors at the same time.
- A mobile phone keeping track of the remaining battery level while letting the user do other tasks.

Hidden/Concealed:

- Cellphone applications that share the same resources, such as multiple photo applications.

- An email-client, when you're waiting for an email to arrive.
- Google Drive Sync, that keeps new files sync with Google Drive.