Spec (Assignment 1)

This assignment is worth 15% and is due before COB of Friday Week 7 (15th July, 18:00 Sydney Time).

This assignment is to be done in pairs. Individual submissions will be accepted but are highly discouraged.

Problem Statement

Consider a finite counter c that is updated by a single writer process and read by $R > 0$ reader processes. The counter consists of some parameter $B > 1$ of bytes, meaning that c ranges over $0 \ldots 2^{8B} - 1$. Both reads and writes of individual bytes are atomic operations. Unfortunately, there are no other concurrency primitives available. No semaphores, no locks, no message passing, no monitors, no transactional memory.

With each write, the counter is incremented by one, wrapping around at the end, so the effect of a write could be expressed as $c := (c + 1 \bmod 2^{8B})$. Readers have the difficult task to read a correct counter value by performing reads of individual counter bytes. A counter value v read by a reader r is considered *correct* if v is actually assumed by c during the time interval while the counter is being read by r.

No process should ever block anyone from doing anything.

Disclaimer and Limitation of Liability

A reader being so slow during a single read that the counter goes through the full sequence of possible counter values ($0 \ldots 2^{8B} - 1$) may return any value. Having said that, this escape clause is not to be abused to implement readers poorly (e.g. by waiting for the most significant byte to go through a full cycle). Readers should make every attempt to get hold of a correct value as soon as possible. Thus we require that, under weak fairness, reads complete eventually even if writes subside.

Task

Solve the problem stated above by designing a shared-variable concurrent algorithm. State your assumptions precisely. Argue why your assumptions are necessary. Implement your solution in Java using Java threads and **no other concurrency**

constructs (e.g. no semaphores, monitor constructs, no synchronized methods, no `java.util.concurrent.atomic`, no nothing). Encode the core algorithm and relevant requirements in Promela so that you can run simulations and maybe even verify that the requirements are satisfied for sufficiently small values of R and B. Using formal and semi-formal reasoning, demonstrate the correctness of your algorithm. Write a compelling report on all that you've done.

Counter with Java

Design and implement a simulation of your algorithmic solution. It should take **three** command line arguments to determine the number R, the number B and the number k of rounds (of writing, resp. reading) each process should perform. Use $k = 0$ as a special case to loop indefinitely. Each process should be modelled as a separate thread.

You cannot use any synchronization primitives such as monitors, synchronized methods, locks, semaphores etc. There's one exception: you **can**, and probably should, declare shared variables as `volatile`. Use neither timers nor time-outs to artificially slow down processes.

Don't worry about weak memory. That is, you may write your program as if reads and writes to e.g. array cells was sequentially consistent.

Counter with Promela

Model your solution with two `#define` statements for the two parameters R and B. Attempt to verify that your model satisfies requirements for small parameter values. It suffices to model the case where all processes loop infinitely ($k = 0$ in the previous task).

Document your work

First of all, make it worth reading — we are neither counters nor robots. Begin with a short summary of what you've achieved. Then, elaborate on your achievements, e.g. by explaining how your algorithm works, what limitations you have encountered and implemented. Also explain precisely how to reproduce your findings.

All text should be typeset, preferably with LaTeX. Diagrams may be hand-drawn. Your report must contain the names and student numbers of **both partners** near the top of the document.

Deliverables

Counter.java
: Your Java solution, with a class called `Counter` that contains a `public static void main(String[] args)` method that can be invoked from a compiled binary. This method should accept the three command line arguments documented above. You may, at your option, also submit other Java files. All Java files will be compiled with the command `javac *.java` and executed with commands like `java Counter 3 2 7000` to run a 3 reader, 2 byte counter for 70000 rounds (ensuring a wrap-around). Do not put any classes inside a package, leave them in the implicit top-level package.

counter.pml
: Your Promela model, including any LTL formulae you may use for verification.

counter.pdf
: Your nicely typeset document containing, at the very least, your student numbers.

partner.txt
: A file with a single line on the format `z<digits>`. This must be the zID of your group partner. For individual submissions, write your own zID.

Submission

Only one member of the pair should submit. Work out who will submit ahead of time. Duplicate submissions are extremely annoying to sort out!

Submit your work by typing:

```
give cs3151 assn1 Counter.java counter.pml counter.pdf partner.txt
```

Add any other Java files you wish to submit to that line.

Free Advice

I caution you against assigning one group member to write Java code, and the other to write Promela code, without first **together** devising a solution to the stated problem. The purpose of the partnership is to provide you with someone with whom you can discuss candidate solutions and derive a solution together. This assignment is not easily compartmentalised into separate tasks, at least not before a solution is found.

Coming up with process descriptions for the writer and the readers may prove to be elusive, at least for a few days of thinking about this. As a first step, try solve the problem for a counter that does not wrap around.

Hint: To save you some time, here's something that does not work at all for any of the processes: write (read) the B bytes in a fixed order, once. Three choices were made: using just B bytes to represent the counter, reading and writing them in the same fixed order, and reading/writing each byte just once. Perhaps reconsider them all.

Hint: This is meant to be a challenge, and we mark it accordingly. Incorrect solutions don't give full marks obviously, but we do award partial marks based on how promising the attempt is and how well the attempt is written, presented and understood.