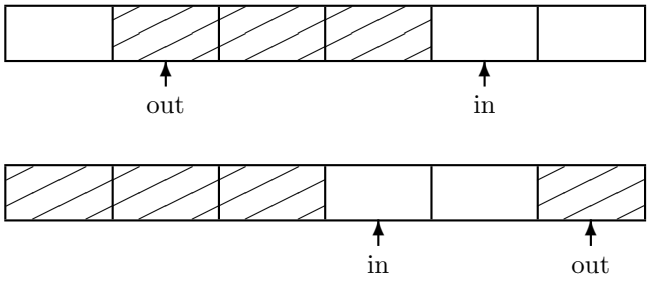
Assignment 1 CSSE7610

Answer questions 1 to 3 below. This assignment is worth 25% of your final mark. It is to be completed individually, and you are required to read and understand the School Statement on Misconduct, available on the School's website at: http://www.itee.uq.edu.au/itee-student-misconduct-including-plagiarism

Due date and time: Friday 30 August, 5pm

1. A bounded buffer is frequently implemented as a circular buffer, which is an array that is indexed modulo its length:



One variable, in, contains the index of the first empty space and another, out, the index of the first full space (if any). If $in > out$, there is data in buffer[out..in-1]; if $in < out$, there is data in buffer[out..N-1] and buffer[0..in-1]; if $in = out$, the buffer is empty. Consider the following algorithm for two processes sharing a circular buffer:

Circular buffer			
dataType array [0..N-1] buffer integer in, out ← 0			
p		**q**	
	dataType d		dataType d
	loop forever		loop forever
p1:	d ← getItem()	q1:	await in != out
p2:	await out != (in+1) mod N	q2:	d ← buffer[out]
p3:	buffer[in] ← d	q3:	out ← (out+1) mod N
p4:	in ← (in+1) mod N	q4:	useItem(d)

Assume that useItem(d) always runs to completion, but that getItem() may not terminate since there may be no items available.

(a) The algorithm is intended to provide mutually exclusive access to individual elements of the array. That is, when p is able to write a value to the array, q should not be able to read from the same array index. State this property as an invariant and prove it using induction. You may need to prove other invariants to do this.

(b) The algorithm is also intended to provide freedom from starvation for each process. That is, after getting an item process p should eventually write it to the buffer, and when an item is in the buffer process q should eventually read it. State these properties using temporal logic and prove they are correct. You may need to prove further invariants to do this.

Deliverable: A file circular.pdf containing your answers to (a) and (b), and your name and student number.

2. Check the above algorithm for any lines which contain more than one critical reference.

   (a) Write a Promela specification for the algorithm that does not have more than one critical reference in any atomic statement. Note that the modulo operator in Promela is % (as in C and Java).

   (b) Use Spin to prove that the algorithm is correct: use assertions to prove mutual exclusion, and an LTL property to prove freedom from starvation. You may need to introduce additional (auxiliary) variables to do this.

   Deliverable: A file circular.pml containing the Promela specification, and a comment describing any changes you made to avoid lines with more than one critical reference, the properties you proved, and your name and student number.

3. Write a Java program to format an arbitrary text file to have exactly 80 characters per line (except for the last line which may have 80 or less characters), after replacing all occurrences of tabs and two or more consecutive spaces with a single space. Your program must use three threads running concurrently. The first thread reads characters from the input file using the provided class A1Reader, and replaces end-of-line characters with spaces. The second thread removes and replaces tabs and removes consecutive occurrences of spaces, and the third thread writes lines of 80 characters to the output file using the provided class A1Writer. The threads must communicate characters via circular buffers of length 20 characters using the algorithm from question 1.

The three threads should be started by the main thread of your Java program. Your program should use *cooperative multitasking*, i.e., a thread should allow others to proceed when it can do no useful work **but not otherwise**, and it should *terminate gracefully*, i.e., all threads should reach the end of their run methods.

Deliverables: A zip file containing a file FileConverter.java with your main method for the program, along with all supporting source (.java) files (apart from A1Reader and A1Writer), and a file readme.txt describing (in a few paragraphs) the approach you have taken to coding your program and providing a list of all your classes and their roles. All files should be well-documented and contain your name and student number (as a comment).

Important: For testing purposes, it is a requirement that you use the provided A1Reader and A1Writer classes. It is also important that you do not modify the provided classes. Also, do not make your submitted files dependent on being in a particular package. That is, remove any lines:

package *packageName*;

Marking criteria

Marks will be given for the correctness and readability of answers to questions 1 to 3 as follows.

Question 1 (10 marks)

- Proof of mutual exclusion (5 marks)
- Proofs of starvation freedom (5 marks)

Question 2 (5 marks)

- Promela specification of algorithm (3 marks)
- Proof method for mutual exclusion (1 mark)
- Proof method for starvation freedom (1 mark)

Question 3 (10 marks)

- Java classes modelling threads and buffers (7 marks)
- Program producing correct behaviour (2 mark)
- readme file (1 mark)