Assignment 1: Mutual exclusion 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: `https://eecs.uq.edu.au/current-students/guidelines-and-policies-students/student-conduct`

Due date and time: Thursday 7 September, 3pm

Peterson's mutual exclusion algorithm is based on Dekker's algorithm but is more concise because it collapses two await statements into one with a compound condition.

Peterson's algorithm			
boolean wantp ← false, wantq ← false integer last ← 1			
p		**q**	
	loop forever		loop forever
p1:	non-critical section	q1:	non-critical section
p2:	wantp ← true	q2:	wantq ← true
p3:	last ← 1	q3:	last ← 2
p4:	await wantq = false or last = 2	q4:	await wantp = false or last = 1
p5:	critical section	q5:	critical section
p6:	wantp ← false	q6:	wantq ← false

1. Prove the correctness of Peterson's algorithm using deductive techniques. Given the following invariants similar to those of Dekker's algorithm

   (a) $(last = 1) \vee (last = 2)$

   (b) $p3..6 \Leftrightarrow wantp$

   (c) $q3..6 \Leftrightarrow wantq$

   first show that

   $(p4 \wedge q5) \Rightarrow (wantq \wedge last = 1)$,
   $(p5 \wedge q4) \Rightarrow (wantp \wedge last = 2)$

   are also invariants, and then use them to prove mutual exclusion holds. Then state the condition for freedom from starvation for process p and provide a proof that it holds.

   Deliverable: A file peterson.pdf containing the correctness arguments, and your name and student number.

2. (a) Write a Promela specification for a modified version of Peterson's algorithm that does not have more than one critical reference in any atomic statement.

   (b) Use Spin to prove that the algorithm is still correct: use an assertion to prove mutual exclusion, and an LTL property to prove freedom from starvation.

   Deliverables: A file peterson.pml containing the Promela specification and including comments detailing how you carried out the proofs and stating any LTL properties required. The pml file must include your name and student number (as a comment).

3. Speed is a card game where the players simultaneously try to get rid of all their cards.

   In this version of the game, we have 2 players and use a pack of 60 cards. Each card has between 1 and 5 motifs of a particular shape and colour on it. For example, a card may have 4 green diamonds, or it may have 5 blue circles, or it may have 1 red square. We will assume there are 6 different shapes and 6 different colours, and that the pack has a random selection of 60 such cards which may contain duplicate cards.

   Each player is dealt a pile of 29 cards and the remaining two cards are placed face up in front of the players. The players pick up the top 3 cards from their pile and must simultaneously try to place them on one of the cards in the centre by either matching the colour, shape or number (of shapes). For example, a player may put a card with 4 red circles on a card with 2 red squares (colour is matched), or a card with 3 green diamonds on a card with 3 blue circles (number is matched). When a player places a card in the centre, it replaces the one it is on top of as one of the two cards that must be matched.

   A player can pick up the next card from their pile whenever they have less than 3 cards in their hand. The winner is the first player to put down all 29 of their cards. If both players reach a point where they are not able to put down any more cards, the game is a draw.

   Write a Java program to simulate the game of Speed. Each player must be implemented as a thread and Peterson's algorithm must be used where mutual exclusion is required. (You should **not** use locks, semaphores, or the **synchronized** or **wait/notify** capabilities of Java objects in this assignment.[1]) The program should use the provided class Card.java for modelling cards, and produce output by calling the appropriate methods of the provided class Event.java. *For testing purposes, it is a requirement that you call the Event class every time one of the events occurs. It is also important that you do not modify either of the provided classes.*

   The critical sections must include the minimal number of statements to ensure the correct behaviour. For example, a player thread should not search for a matching card while in the critical section, although it may need to check that a previously chosen card is still a valid choice. As well as global variables representing the cards in the centre of play, you may require additional global variables, e.g. to allow the threads to determine a draw has occurred.

   Deliverables: A **zip** file containing the file Speed.java (which includes your main method) along with all supporting source (.java) files (apart from Card and Event), and a file readme.txt describing (in a few paragraphs) the approach you have taken to coding the program and providing a list of all your classes and their roles. All Java files should be well-documented and in particular the critical section code and code for detecting draws should be well explained. All files should also contain your name and student number (as a comment).

   To assist with my testing of your Java code. Please do not make your submitted files dependent on being in a particular package. That is, remove any lines:

   package *packageName*;

[1] Of course, you would use such constructs in practice, and you will do so in Assignment 2. The intention of this assignment is to give you experience with mutual exclusion algorithms.

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 invariants (4 marks)
- Proof of mutual exclusion using invariants (2 marks)
- Proof of starvation freedom (4 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)

- readme file (1 mark)
- Java thread class modelling players (5 marks)
- Program producing correct behaviour for a winning game (2 marks)
- Program producing correct behaviour for a draw (2 marks)