Concurrent Programming

Exercise Booklet 2: Mutual Exclusion

1. Consider the following proposal for solving the mutual exclusion problem for n processes, that uses the following functions and shared variables:

```
global int current = 0, turns = 0;

RequestTurn() {
  turn  = turns;
  turns = turns + 1;
  return turn;
}

FreeTurn() {
  current = current + 1;
  turns = turns - 1;
}
```

We assume that each thread executes the following protocol:

```
// non-critical section
turn = RequestTurn();
while (current != turn);
// critical section
FreeTurn();
// non-critical section
```

Show that this proposal does not solve the MEP. Indicate clearly which condition/s are not satisfied and illustrate by means of a trace.

2. Build a trace that shows that attempt IV at solving the MEP, as seen in class, does not enjoy freedom from starvation.

3. Consider the following extension of Peterson's algorithm for n processes ($n > 2$) that uses the following shared variables:

```
global boolean[] flags = replicate(n,false);
global int turn = 0;
```

Moreover, each thread is identified by the value of the local variable `threadId` (which takes values between 0 and $n-1$) . Each thread uses the following protocol.

```
...
// non-critical section
flags[threadId] = true;
turn = (turn+1) % n;
while (flags[other] && turn==other);
// critical section
flags[threadid] = false;
// non-critical section
...
```

Show that this proposal does not solve the MEP. Indicate clearly which condition/s are not satisfied and illustrate by means of a trace.

4. Consider the following extension of Peterson's algorithm for n processes ($n > 2$) that uses the following shared variables:

```
global boolean[] flags = replicate(n,false);
```

and the following auxiliary function

```
boolean SomeOtherTrue(id) {
  result = false;
  for (i : range(0,n-1))
    if (i != id)
      result = result || flags[i];
  return result;
}
```

Moreover, each thread is identifierd by the value of the local variable `threadId` (which takes values between 0 and $n-1$). Each thread uses the following protocol.

```
  ...
  // non-critical section
  flags[threadId] = true;
  while (SomeOtherTrue(threadId));
  // critical section
  flags[threadid] = false;
  // non-critical section
  ...
```

Show that this proposal does not solve the MEP. Indicate clearly which condition/s are not satisfied and illustrate by means of a trace.

5. Use a state diagram to show that Peterson's algorithm solves the MEP.

6. Given *Bakery's Algorithm*, show that the condition `ticket`[j] < `ticket`[`threadId`] in the second while is necessary. In other words, show that the algorithm that is obtained by removing this condition (depicted below) fails to solve the MEP. Indicate clearly which condition/s are not satisfied and illustrate by means of a trace.

```
global boolean[] choosing = replicate(N,false);
global int[] ticket = replicate(N,0);

thread {
  // non-critical section
  choosing[threadId] = true;
  ticket[threadId] = 1 + maximum(ticket);
  choosing[threadId] = false;
  for (j : range(0,n-1)) {
    while (choosing[j]);
    while (ticket[j] != 0 &&
           (ticket[j] < ticket[threadId] ||
           (ticket[j] == ticket[threadId])));
  }
  // critical section
  ticket[threadId] = 0;
  // non-critical section
}
```