National University of Singapore
School of Computing

CS2106 **Tutorial 3** Semester 1 11/12

This tutorial contains a ⋆ question. The ⋆ questions (i) are challenging, (ii) are beyond the scope of CS2106, and (iii) will be not discussed during tutorial. Interested students may however email the solutions to the lecturer. Students who solve three ⋆ questions correctly will earn a coffee and enter CS2106 Hall of Fame.

1. To ensure mutual exclusion of a critical region between two processes, Process 0 and Process 1, the following implementation of `enter()` and `leave()` are called before entering and leaving a critical region respectively. Before entering a critical region, Process 0 calls `enter(0, 1)`, while Process 1 calls `enter(1, 0)`. When leaving the critical region, Process 0 calls `leave(0)`, while Process 1 calls `leave(1)`. Both `interest[0]` and `interest[1]` are set to 0 initially.

```
void enter(int process, int other)
{
        interest[process] = 1;
        while (interest[other] == 1);
}

void leave(int process)
{
        interest[process] = 0;
}
```

   (a) Does the implementation above properly ensures *mutual exclusion*, i.e., no two process can enter the critical region simultaneously?
   (b) Does the implementation above properly ensures *progress*, i.e., both processes will not enter the loop and wait forever?

2. Argue how Peterson's algorithm ensures the following:
   (a) *mutual exclusion*, i.e., no two processes can enter the critical region simultaneously.
   (b) *progress*, i.e., both processes will not enter the loop and wait forever.
   (c) *bounded waiting*, i.e., no one process will dominate the entrance into critical region (and the other process waits forever).

3. On a machine that does not support the TSL instruction, the following C equivalent of `test-and-set` has been implemented:

```
        int test_and_set(int *lock)
        {
                if (*lock) {
                        return 1;
                } else {
                        *lock = 1;
                        return 0;
                }
        }
```

   To use it in `enter()` and `leave()` functions:

```
        void enter() {
                while (test_and_set(&lock));
        }
        void leave() {
                lock = 0;
        }
```

(a) Since the above software implementation of `test_and_set` is not atomic, the implementation of `enter()` and `leave()` does not ensure mutual exclusion. Explain why.

(b) Suggest steps that the operating system can do before and after calling `test_and_set` to make this implementation atomic.

4. ($\star$) Extend Peterson's algorithm for mutual exclusion to 3 processes. Proof that your solution correctly ensures mutual exclusion, progress, and bounded waiting.