Midterm Examination
CSc 422, Fall 2020

October 8, 2020

Instructions: Please put your name in the upper right hand corner of the page. The exam consists of 5 problems on 6 pages and is open notes and book (but closed neighbor). The point values of the problems vary; please take this into account when allocating your time. The exam is worth 100 points.

Be concise and indicate clearly what your answer is. Presentation and simplicity of your answers may affect your grade.

Problem 1: Warmup (15 points [3 each])

Do **not** explain your answers.

A. True/False: A thread in a critical section cannot be context switched.

B. True/False: A `P` operation on a semaphore S may block in the case that the previous operation on S was a `V` operation.

C. True/False: A `V` operation on a semaphore S may increment the value of S in the case that the previous operation on S was a `P` operation.

D. True/False: A thread need not acquire Lock L before releasing Lock L.

E. Suppose a program has only a `co` statement has two arms, which take 1 and 2 seconds, respectively. If this program is executed on a two-core machine, what is its worst-case execution time? Assume that you know do not know how the `co` statement is implemented.

Problem 2: Concurrency (20 points)

Given the following code (lines are numbered for convenience):

```
int x = 0, z = 0, w = 0, a = 0, b = 0
Lock L

(1)  co
(2)    Acquire(L); b = b + 3; Release(L)
(3)    w = w + 2
(4)    co
(5)      x = x + 1; a = a + 4
(6)    //
(7)      Acquire(L); x = x + 2; Release(L)
(8)    oc
(9)  //
(10)   z = z + 1
(11)   co
(12)     z = z + 2
(13)   //
(14)     w = w + 1; a = a + 3; Acquire(L); b = b + 2; Release(L)
(15)   oc
(16) oc
```


What are the potential final values for each variable? Explain clearly and concisely; you need not list all five-tuples—just answer for each variable individually. Recall that statements of the form $y = y + k$ are *not* atomic. Please note that this program has one (outer) co statement, and each arm of the (outer) co statement itself contains a (nested) co statement.

Problem 3: Bounded Buffers (25 points [6/6/6/7])

Consider the semaphore solution to (general) Bounded Buffer problem discussed in class and reproduced below. There are an arbitrary number of producers and consumers. Note that "+" indicates modular arithmetic, and that *empty*, *full*, *mutexP*, and *mutexC* are semaphores, with initial values given.

```
int buffer[n], front = 0, rear = 0;
sem empty := n, full := 0, mutexP := 1, mutexC := 1;

Producer(item) {
    P(empty); P(mutexP);
    buf[rear] = item;
    rear = rear+1;
    V(mutexP); V(full);
}

Consumer() {
    P(full); P(mutexC);
    item = buf[front];
    front = front+1;
    V(mutexC); V(empty);
    return item;
}
```


For (A) through (C), what effect (if any) would making each of the following changes, *independently*, have on the solution? Be specific in your answers, using a one- or two-sentence justification. Vague answers will receive little, if any, credit.

A. Interchanging P(empty) and P(mutexP) in the producer *and* P(full) and P(mutexC) in the consumer.

B. Initializing empty to $n - 1$.

C. Changing V(empty) to V(full) in the consumer.

D. If front = rear, what relationship must hold between empty and full? Explain.

Problem 4: Mistakes with Locks (20 points [7/7/6])

For each of the following parts, assume function `foo` is executed by multiple threads and that mutual exclusion is required between those threads. In each part, explain clearly and concisely what mistake(s) the programmer has made (there may be more than one error in a given part). Assume that there are no accesses to x, y, or z from anywhere else in the code. Also, function `GetInput` reads an arbitrary (i.e., unknown to you) integer from the command line.

A.

```
Lock L
int x = 0, y = GetInput(), z = GetInput()

foo() {   // foo needs mutual exclusion between threads
  Acquire(L)
  if (y == 3) {
    x = x + 1
    Release (L)
  }
  if (z == 6) {
    Release (L)
  }
  Release(L)
  return
}
```


B.

```
Lock L

foo(int y) {   // foo needs mutual exclusion between threads
  Acquire(L)
  if (y == 0) { // base case
    return
  }
  else
    foo(y-1)
  }
  Release(L)
  return
}
```

C.

```
Lock L1, L2
int x = GetInput(), y = 0, z = 0

foo(int myThreadId) {   // myThreadId is unique per thread
  if (x == myThreadId) {
    Acquire(L1)
    Acquire(L2)
    y = y + 2
    z = z + 3
  }
  else {
    Acquire(L2)
    Acquire(L1)
    z = z + 2
    y = y + 3
  }
  Release(L1)
  Release(L2)
}
```

Problem 5: Parallel Programming (20 points [10 each])

A. A given sequential program has 100 seconds of computation that is inherently sequential, and it *additionally* has 500 seconds of computation that can be parallelized. What is the (theoretical) maximum speedup this program can achieve? Explain briefly.

B. Assume the same 100 seconds of inherently sequential computation, and now assume that the additional 500 seconds of computation that can be parallelized is split over four threads, with ids 1, 2, 3, and 4. Assume that each thread is run on a unique core. The execution time of thread i (in seconds) is governed by the following function: $T(i) = 50 \times i$. Assume there is no overhead for thread creation or thread synchronization. What is the speedup this program will achieve? Explain briefly.