Selected Topics in Distributed Computing

Final exam

January 17, 2008

Last Name:

First Name:

Problem	Max points	Points obtained
1	2	
2	2	
3	2	
4	1	
5	1	
6	2	
Total	10	

Exam rules:

1. Exam time: from 14.15 to 17.15.
2. The exam is closed book. No electronic devices are allowed.
3. You can use any notation for algorithms, but remember to write which variables represent shared objects (e.g., registers) and which are process-local.
4. Describe shortly the main idea behind every algorithm you give.
5. Keep in mind that only one operation on one shared object (e.g., a read *or* a write of a register) can be executed by a process in a single step. To avoid confusion (and common mistakes):
   - make your algorithms access registers only by explicitly calling register operations *read* and *write* (e.g., use `R.write(5)` instead of `R := 5`), and
   - write only a single atomic step in each line of an algorithm.
6. The exam grade will be computed in the following way: 1.0 (for handing in the exam) plus the number of points obtained divided by 2.

Assumption: We assume an asynchronous, shared-memory system of n processes, out of which $n - 1$ might crash.

Good luck!

Problem 1 (2 points)

Write an algorithm that implements a MRMW atomic multi-valued wait-free register using (any number of) MRSW atomic multi-valued wait-free registers.

Problem 2 (2 points)

Write an algorithm that implements an atomic wait-free counter object using only (MRMW atomic multi-valued wait-free) registers.

Reminder: A *counter* object has two operations: *inc()* and *read()*, and maintains an integer x initialized to 0. The sequential specification of a conter object is the following:

```
read():
  return x;

inc():
  x := x + 1;
  return ok;
```

Problem 3 (2 points)

Task A (0.5 point). Give the specification of a wait-free consensus object.

Task B (1.5 point). A *swap* object has one operation *swap(v)*, and maintains an integer x initialized to 0. Intuitively, the *swap(v)* operation (with value v given as an argument) atomically changes the value of x to v and returns the previous value of x (i.e., swaps the values of x and v). More precisely, the sequential specification of a *swap* object is as follows:

```
swap(v):
  tmp := x;
  x := v;
  return tmp;
```

Write an implementation of a wait-free consensus object using only (atomic, wait-free) swap objects and (MRMW atomic multi-valued wait-free) registers, in a system with *two* processes (i.e., for $n = 2$).

Problem 4 (1 point)

Consider the following, *incorrect*, implementation of an anonymous obstruction-free consensus object from counters:

uses: C_0, C_1 – counters

upon $propose(v)$ **do**
  while *true* **do**
    $(x_0, x_1) \leftarrow readCounters()$
    if $x_0 > x_1$ **then** $v \leftarrow 0$
    else if $x_1 > x_0$ **then** $v \leftarrow 1$
    if $|x_0 - x_1| \geq 1$ **then return** v
    $C_v.inc()$

where *readCounters* procedure is implemented as follows:

upon $readCounters()$ **do**
  while *true* **do**
    $x_0 \leftarrow C_0.read()$
    $x_1 \leftarrow C_1.read()$
    $x_0' \leftarrow C_0.read()$
    if $x_0 = x_0'$ **then return** (x_0, x_1)

Give an execution of the above algorithm that shows that the algorithm is not a correct implementation of an anonymous obstruction-free consensus object, i.e., an execution in which some property of obstruction-free consensus is violated.

Problem 5 (1 point)

Consider a simple (distributed) *memory allocation* object that has only one operation called *dmalloc*. If a process p_i invokes *dmalloc*() (with no parameters), p_i is returned an address of a free memory block that p_i can subsequently use. For simplicity, we assume that:

1. All memory blocks are of the same size, and their addresses are integers 0, 1, 2, . . . , and
2. No process invokes *dmalloc* more than M times in any execution, where M is some known constant.

The memory allocation object ensures the following (in every execution):

1. No two processes are returned the same address by *dmalloc*.
2. The highest address w returned by *dmalloc* (at any process) in a given execution is bounded by a function $f(k)$, where k is the number of invocations of *dmalloc* in that execution, and f is independent of the total number of processes n.

Write an algorithm that implements a wait-free memory allocation object (i.e., its *dmalloc* operation) using only (MRMW atomic multi-valued wait-free) registers.

Problem 6 (2 points)

Intuitively, *"fail-only"-consensus* provides an implementation of consensus, but allows some *propose* operations to *abort* when these cannot terminate and return a decision value because of other concurrent invocations of *propose*. When *propose* aborts, it means that the operation did not take place, and so the value proposed using this operation has not been "registered" by the consensus object. A process which *propose* operation has been aborted may retry the operation many times (possibly with different proposed value), until a decision value is returned.

More precisely, let D be any set, such that $\perp \notin D$. A "fail-only"-consensus object implements a single operation, called *propose*, that takes a value $v \in D$ as an argument and returns a value $v' \in D \cup \{\perp\}$. If a process p_i is returned a non-$\perp$ value v' from *propose*(v_i), we say that p_i *decides* value v'. Once p_i decides some value, p_i does not invoke *propose* anymore. We say that p_i *commits* value v if p_i invokes *propose*(v) and decides value v (i.e., p_i decides the value that p_i has proposed). When operation *propose* returns $\perp$, we say that the operation *aborts*. We say that a *propose* operation executed by a process p_i is *step contention-free*, if no process other thatn p_i executes a step between the invocation and the response of the operation. Every "fail-only"-consensus object satisfies the following properties in every execution::

Fo-validity If some process decides value v, then some process commits v.

Agreement No two processes decide different values.

Fo-obstruction-freedom If a *propose* operation is step contention-free, then the operation cannot abort.

Wait-freedom If a correct process p_i invokes a *propose* operation, then p_i eventually returns from the operation.

The *consensus number* of a shared object O is the maximum number of processes among which one can solve consensus using any number of instances of O (i.e., objects of the same type as O) and (MRMW atomic multi-valued wait-free) registers.

Prove that the consensus number of a "fail-only"-consensus object equals 2 by:

1. Showing an algorithm that implements a wait-free 2-consensus object (i.e., consensus for 2 processes) using only "fail-only"-consensus objects and (MRMW atomic multi-valued wait-free) registers. **(1 point)**

2. Proving that it is impossible to implement a wait-free 3-consensus object using only "fail-only"-consensus objects and (MRMW atomic multi-valued wait-free) registers. **(1 point)**