CS 5800 Distributed OS - Spring 2017
Homework 2

Josh Herman
Dalton Cole
Neil Blood

February 23, 2017

Problem 1 Suppose there are two GTAs (Alice and Bob) for CS384, each of whom is responsible for one section. In other words, Alice works as a grader for section 1 (S_1), and Bob works as a grader for section 2 (S_2). At the end of the semester, final grades will be assigned to students in the class based on their final total points. Assume there are two possible grades A and B, and Alice and Bob know the final total points for every student in their own sections (Alice does not know the final total points of students in S_2, and vice versa).

You can assume that $|S_1| = |S_2| = n$ and $S_1 \cap S_2 = \emptyset$. Let $n = 3, S_1 = \{s_{11}(91); s_{12}(92); s_{13}(80)\}$ and $S_2 = \{s_{21}(81); s_{22}(82); s_{23}(97)\}$, where the value in the parentheses is the final total points for each student. At the end of your protocol, Alice will give A to s_{11}, s_{12} and B to $s13$. Similarly, Bob will give A to s_{23} and B to s_{21}, s_{22}.

1. Design a distributed algorithm that assigns A to the upper half of all students and B to the lower half of all students in the class. Additionally, please make sure to minimize the communication cost of your protocol.

   With the function $getMedian(A, B)$ Alice and Bob could figure out the overall median with a minimal communication cost and then both Alice and Bob could go through their section's grades and assign A's to those above the overall median and B's to those below the overall median.

$getMedian(A, B)$
{
    $subA = A$
    $subB = B$
    $while(size(subA) > 2)$
    {
        $mA = median(subA)$
        $mB = median(subB)$
        Alice send mA to Bob
        Bob send mB to Alice
        $if\ mA == mB:$
            return median is mA
        $else\ if\ mA > mB$
            $subA = [subA[0]...mA]$
            $subB = [mB...subB[last]]$
        $else\ if\ mA < mB$
            $subA = [mA...subA[last]]$
            $subB = [subB[0]...mB]$
    }
    Alice sends subA[0] *and subA*[1] *to Bob*
    Bob computes : $median = \frac{\max(subA[0], subB[0]) + \min(subA[1], subB[1])}{2}$
    Bob sends median to Alice
    return median
}

2. Analyze the complexity of the algorithm in terms of the number of rounds, the size of the message for each round (assume 2 bytes are sufficient to store a final total point), and the local or individual computational costs.

   The number of rounds for the algorithm, the while statement, would take $\log(n)$ steps and there would be two messages being sent each round, totaling 4 bytes. At the end of the while loop there would need to be two messages sent by Alice, totaling 4 bytes. So added all together the algorithm would cost $4\log(n)+4$ bytes. The computational cost would be $O(\log(n))$ since that's the number of times the while loop would iterate.

Problem 2 Define a well-founded set to prove that the program terminates in a bounded number of steps. You can assume that t is indeed the value of one of the elements in the array X.

The program is guaranteed to finish in n steps. This is because, in the worst case, $x[n-1] = t$. This is true because t is guaranteed to be in the set. In other words:
Set $s = \{\}$
The max size of s will be n. For every $x[i] \neq t$, append $x[i]$ to s. This will

represent all of the elements before t in the list. The max size of the set is n, which is the maximum number of steps to terminate the program.

Problem 3 Describe the desired safety and liveness properties when we use Google Drive to share files.

Safety: When two or more users edit the same document at the same time, their edits are never lost.
Liveliness: Document will auto-save eventually, saving the data.

Problem 4 Refer to slide #19 of "handout4.pdf", prove a different example (from the one given on slide # 21) to prove that the program does not guarantee termination.

State	Action	c[0]	c[1]	c[2]	c[3]
A		0	0	2	0
B	P_0 moves	2	0	2	0
C	P_0 moves	0	0	2	0

As can be seen from the above table, states A and C are identical and the program returned to its initial state, thus the program can cycle with this given input, thus not guaranteeing termination.

Problem 5 Refer to slide #30 of "handout4.pdf", modify the program in a way that the safety property is still preserved, but the mathematical representation of the invariant is different from the one given on slide #31. Using induction to prove the safety property holds.

define $c1, c2 : channel$
initially $c1 = c2 = \emptyset$
$\{program for \mathbf{T}\}$
define$t : integer(initially\ t = 6)$
1 **do** $t > 0$ $\rightarrow$ send message along $c1; t := t - 1$
2 $\neg empty(c2)$ $\rightarrow$ receive message from $c2; t := t + 1$
od
$\{program for \mathbf{R}\}$
define$r : integer(initially\ r = 6)$
3 **do** $\neg empty(c1)$ $\rightarrow$ receive message from $c1; r := r + 1$
4 $r > 0$ $\rightarrow$ send message along $c2; r := r - 1$
od
Invariant changes to: $I \equiv (t \geq 0) \wedge (r \geq 0) \wedge (n1 + t + n2 + r = 12)$

Prove by induction that the safety property holds:
Base Case:
Initially: $n1 = n2 = 0, t = r = 6 \Rightarrow n1 + t + n2 + r = 12$
Induction Step: assume that I holds in the current state. We need to show that I holds after he execution of every eligible guarded action.

1. **After action 1**: The values of $(t + n1), n2, r$ remain unchanged. Also, since the guard is true when $t > 0$ and the action decrements t by 1, the condition $t \geq 0$ holds. Thus, I holds.

2. **After action 2**: The values of $(t + n2), n1, r$ remain unchanged. Also, since the value of t can only be incremented, so $t \geq 0$ holds. Thus, I holds.

3. **After action 3**: The values of $(r + n1), n2, t$ remain unchanged. Also, since the value of r can only be incremented, so $r \geq 0$ holds. Thus, I holds.

4. **After action 4**: The values of $(r + n2), n1, rt$ remain unchanged. Also, since the guard is true when $r > 0$ and the action decrements r by 1, the condition $r \geq 0$ holds. Thus, I holds.

Therefore, with proof by induction the safety property still holds.

Problem 6 Show how to use the clock synchronization program (slide #30 of "handout4.pdf") to sync the clocks given in Figure 1.

2	0	1	0	2
1	2	2	2	1
0	0	0	0	0