Assignment 9

Problem Statement Consider a variant of the mutual exclusion problem which satisfies the following two properties:

1. At most two processes are in the critical section at the same time.
2. If there is a process in the trying section and at most one process in the critical section, then some process enters the critical section without waiting.

P1. Give an algorithm for this problem using any objects of bounded size. You may use objects what were not defined in class provided they are clearly defined. Prove the algorithm is correct.

Algorithm. We will make use of the Read-Modify-Write registers as described on page 66 of the Attiya text book. In particular, a Read-Modify-Write register R has one operation rmw which takes as input the register R and a function f. In one atomic operation the process reads in the current value v of R, computes $f(v)$, and writes $f(v)$ to R. The original value of v of R is returned. For this problem, let R take on values $\{0, 1, 2\}$ and be initialized to 0. The functions *inc* and *dec* be defined as follows:

$$inc(v) = \begin{cases} 2 & \text{if } v = 2 \\ v+1 & \text{otherwise} \end{cases} \quad \text{and} \quad dec(v) = \begin{cases} 0 & \text{if } v = 0 \\ v-1 & \text{otherwise} \end{cases}$$

A process p_i in the entry section will execute $\mathsf{rmw}(R, inc)$ and write the result into a local variable v_i. If $v_i < 2$ then p_i enters the critical section. Otherwise p_i tries $\mathsf{rmw}(R, inc)$ again. Once p_i enters the exit section it will execute $\mathsf{rmw}(R, dec)$ once, send rem_i, and enter the remainder section. Next we will prove that this algorithm is correct.

Claim 1. *During an admissible execution of the algorithm, at most two processes are in the critical section at the same time.*

Proof. Suppose for a contradiction there is an admissible execution where three or more processes are in the critical section. Let C be the first configuration where this occurs and let p_0, p_1, p_2 be in the critical section. We say that a process is *critical* if it is about to enter the critical section or about to leave the exit section (here a process is critical if it receives a value less than 2 from $\mathsf{rmw}(R, inc)$ or sends rem_i). Thus in the configuration C' before C one of the processes is critical. W.l.o.g suppose p_2 is critical in C'. Note that when p_0 and p_1 entered the critical section, they both incremented R by 1. Since they have not yet performed their exit section code, $R \geq 2$ unless some other process p_j successfully executed $\mathsf{rmw}(R, dec)$. This is a contradiction since C is the first instance with three or more processes in the critical section. □

Claim 2. *If there is a process in the trying section and at most one process in the critical section, then some process enters the critical section without waiting.*

Proof. We will show that the number of processes which have just executed $\mathsf{rmw}(R, inc)$ and received a value less than two, are in the critical section, and are just about to execute $\mathsf{rmw}(R, dec)$ is equal to the value of R. The proof is by induction using the modified mutual exclusion property

above. The property holds at the beginning since R is initialized to 0 and no process is in any of the above states. If process p_i executed $\mathsf{rmw}(R, inc)$ and received $v_i < 2$, then there were fewer than two process in the above three state by the induction hypotheses. p_i's rmw incremented R and p_i is now in one of the three state. If p_i is in one of the three states then $R > 0$ by the inductive hypothesis. After p_i executed $\mathsf{rmw}(R, dec)$, R decreases by one and p_i is no longer in one of the three states. The truth of the claim follows from the truth of the above statement. □

P2. Give an algorithm for this problem using only registers. Prove that your algorithm is correct.

Algorithm. We will extend the bakery algorithm so that at most two processes can be in the critical section at the same time. We use a vector of n binary registers $Choose$ where $Choose[i]$ indicates whether or not process p_i is choosing a ticket. All entries of $Choose$ are initialized to false. We use a vector of n registers $Number$ where $Number[i]$ indicates p_i's ticket. All entries of $Number$ are initialized to 0. Further, process p_i will have a local variable $skip_i$ which keeps track of the index of the process they skipped over. $skip_i$ is initialized to -1.

The general structure of the algorithm is the same. Process p_i sets $Choose[i]$, chooses a number v which is one more than the maximum of $Number$, and sets its ticket to $\langle v, i\rangle$. Next p_i looks at the other processes — starting with the one with the smallest id — waits for them to choose and compares tickets. If they have a larger (in lexical-graphical order) ticket, p_i moves to the next process. If they have a smaller ticket and $skip_i = -1$, then p_i changes $skip_i$ to be the index of the current process and continues. If $skip_i \geq 0$, then p_i must wait on both the current processes and the process at index $skip_i$. See Algorithm 1 for associated pseudo-code.

Claim 3. *During an admissible execution of the algorithm, at most two processes are in the critical section at the same time.*

Proof. Suppose p_i and p_j are in the critical section. We will show that for all processes p_k such that $Number[k] \neq \langle 0, 0\rangle$ (some process which is in the entry section) must satisfy $Number[k] > Number[i]$ and $Number[k] > Number[j]$. W.l.o.g suppose that $Number[i] < Number[j]$. Since process p_j entered the critical section, it must have finished the for loop. In particular, p_j must have waited for p_i to finish choosing a number and observed that $Number[i] < Number[j]$. There are two cases to consider. If $skip_j \geq 0$ then p_j must wait and cannot enter the critical section. Thus $skip_j = -1$ before considering index i then $skip_j = i$ afterwards. If $k < i$ it must be the case that $Number[k] > Number[j]$ since $skip_j \neq k$. Otherwise $k > i$. p_j waited for p_k to choose a number and must have observed that $Number[k] > Number[j]$ before entering the critical section. Thus $Number[k] > Number[j] > Number[i]$ as required.

From the above, when two processes are in the critical section, all processes in the entry section will see two smaller tickets so will not enter the critical section. □

Claim 4. *If there is a process in the trying section and at most one process in the critical section, then some process enters the critical section without waiting.*

Proof. Let p_j be the process in the critical section. Consider the process p_i with the smallest ticket currently in the entry section. We claim that p_i will complete the for loop and enter the critical section. Suppose p_i sees a process p_k ($k \neq j$) such that $Number[j] \neq \langle 0, 0\rangle$ and $Number[j] < Number[i]$. p_k cannot be in the entry section by assumption and p_k cannot be in the critical

Algorithm 1 Implementation of modified mutex only using registers: code for p_i.

1: $\langle ENTRY_i \rangle$
2: $Choose[i] \leftarrow true$
3: $Number[i] \leftarrow \langle \max Number + 1, i \rangle$
4: $Choose[i] \leftarrow false$
5: **for** j from 0 to $n-1$ $(j \neq i)$ **do**
6: wait until $\neg Choose[j]$
7: **if** $skip_i = -1$ **then**
8: **if** $Number[j] \leq Number[i]$ **then**
9: $skip_i = j$
10: **end if**
11: **else**
12: wait until $Number[j] = \langle 0, 0 \rangle$ or $Number[j] > Number[i]$ or
13: $Number[skip_i] = \langle 0, 0 \rangle$ or $Number[skip_i] > Number[i]$
14: **if** $Number[skip_i] = \langle 0, 0 \rangle$ or $Number[skip_i] > Number[i]$ **then**
15: $skip_i = -1$
16: **end if**
17: **end if**
18: **end for**
19: $crit_i$
20:
21: $\langle EXIT_i \rangle$
22: $Number[i] \leftarrow \langle 0, 0 \rangle$
23: $skip_i \leftarrow -1$
24: rem_i

section since p_i is the only process in the critical section. Thus p_k must be in the exit section. In finitely many steps p_k will set $Number[j]$ to be $\langle 0, 0 \rangle$. Thus p_i will only remain in the for-loop for finitely many steps and will enter the critical section. □