6-1-2011

Reader-Writer Exclusion Supporting Upgrade and Downgrade with Starvation Freedom

Matthew Elkherj
Dartmouth College

Dartmouth Computer Science Technical Report TR2011-68

READER-WRITER EXCLUSION SUPPORTING UPGRADE AND DOWNGRADE WITH STARVATION FREEDOM

Matthew Elkherj
Thesis Advisor: Prasad Jayanti

Abstract. In this thesis we give a constant Remote-Memory-Reference (on CC systems) reader-writer exclusion algorithm supporting upgrade and downgrade, built from a reader-writer exclusion algorithm by Jayanti and Liu. The algorithm is starvation-free, and allows for repeated upgrades and downgrades.

1. Introduction

Reader-Writer exclusion, described in [5] and [1], has been a problem of interest to computer scientists from as early as 1971, and has been specified in a variety of ways over these years. Like many other locking problems, an algorithm that solves Reader-Writer exclusion can be broken into a remainder section, doorway, waiting room, critical section, and exit section. The doorway and waiting room are together called the try section. This format is used for the algorithm in [1], and a similar format was even used in Lamport's Bakery algorithm [7]. Processes that don't care to write or read stay in the remainder section, and don't execute the rest of the sections. Processes that want to read/write will try to enter the Critical Section (abbreviated CS), the section where one could imagine the reading/writing of a buffer happening. To do so, a process must first register interest by completing the doorway, wait for its turn in the waiting room, and once, done register its completion in the exit section to allow other processes to enter the CS. A solution to a version of the reader-writer exclusion problem thus involves giving code for the reader-try-section, writer-try-section, reader-exit-section, and writer-exit-section.

The solution in [1] breaks the problem down into three distinct versions: when writers have priority over readers to enter the CS, when readers have priority over writers to enter the CS, and when neither class has priority and we require simply that no process starves. It is easy to intuit why the requirements of writer-priority and reader-priority must give distinct solutions, but why can't an algorithm be both reader/writer-priority and starvation-free? The reason is that, in both the writer-priority case and reader-priority case, it is possible to imagine a lower priority process starving (executing infinitely many steps without getting into the CS) if one higher-priority process after another blocks it. This follows directly from the properties specified in [1]. As a result, starvation-freedom is a completely different requirement than the priority requirements, and requires a separate solution.

Such a solution has been discovered by Jayanti and Liu, and we add to this solution functionality for upgrading and downgrading. The upgrade/downgrade functionality provided allows readers and writers the flexibility to change their mind as many times as desired once in the Critical Section. This means a reader can upgrade, then downgrade, then upgrade, ... Likewise, a writer can execute a downgrade, then upgrade, ... A reader also has the flexibility to attempt upgrading

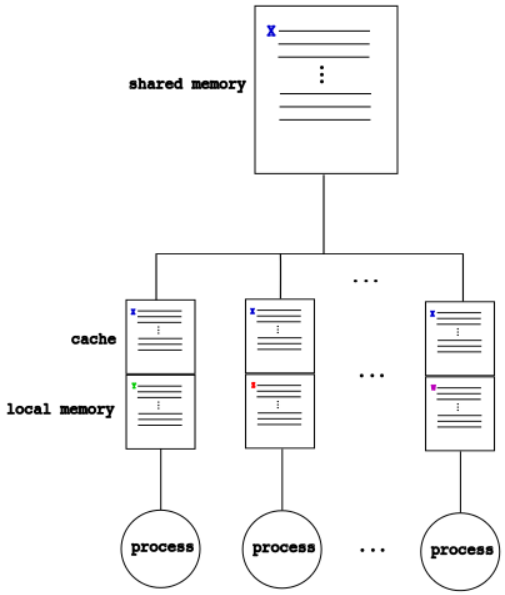

FIGURE 2.1. Cache-Coherent (CC) Model, a shared memory model where writes immediately change the variables in shared memory, but not the caches of other processes

as many times as it wants, assuming it keeps failing. To rigorously specify the problem, we move on to the system model.

2. System Model

We can only strive for an algorithm that satisfies certain properties if we accurately specify the system on which the algorithm will run. The exclusion problem we are looking to solve will certainly require some communication between processes, so we are looking for some distributed system on which to run our algorithm. We can think of interprocess communication as writing into memory locations that other processes can see, and two commonly used systems of this kind are Distributed Shared-Memory (DSM) and Cache-Coherent (CC) systems, as discussed in [2]. It is fruitful to focus on minimizing Remote-Memory-References (RMR) in designing an algorithm, since this is often the major cost incurred on practical distributed systems.

It is impossible to obtain a constant-RMR algorithm on a DSM architecture that satisfies the properties we want of Reader-Writer exclusion [3]. Thus, we only consider Cache-Coherent systems to run our algorithm on.

A Cache-Coherent System is one in which there is shared memory that all processes can access, and each process has a cache of all shared variables and its own local memory to perform local computations. The Cache-Coherent architecture ensures that each processes' local cache of shared variables is consistent in the following manner: when a shared variable X is written, every process' cache of X is invalidated. If another process p wants to read X, a cache miss occurs, and p's cache of X gets a copy of the updated value of X in shared memory. On the other

hand, if a process p is reading a shared variable and no change has been to the variable, it can just read its cache without a miss.

The system we are using must also support some primitive shared-memory objects. We describe each object and its RMR cost per operation:

Register: An $n-$process register object $\mathcal{O}$ allows n processes to read and write to $\mathcal{O}$. A read of register $\mathcal{O}$ into variable x local to some process is denoted $x = \mathcal{O}$. A write to register $\mathcal{O}$ of the value 10 is denoted $\mathcal{O} = 10$. When a process p writes writes to object $\mathcal{O}$, even if it doesn't change the value of $\mathcal{O}$ in writing, a cost 1 RMR is incurred upon p. When a process p reads $\mathcal{O}$, if the value hasn't been written to (even if the write changed nothing!) since the last time it read, then p experiences a cache-miss and is charged 1 RMR. If p reads $\mathcal{O}$ but no process has written to $\mathcal{O}$ since p's last read of $\mathcal{O}$, then p isn't charged an RMR.

Fetch-and-Add(abbreviated F&A): An $n-$process $F\&A$ object $\mathcal{O}$ allows any of n processes to locally store the current value of $\mathcal{O}$ and increment/decrement the value of $\mathcal{O}$ in a single atomic step. A $F\&A$ operation by some process that stores the value of $\mathcal{O}$ into x and increments the value of $\mathcal{O}$ by 12 is denoted $x = F\&A(\mathcal{O}, 12)$. Any $F\&A(\mathcal{O}, c)$ by process p, where $c \neq 0$, is changing the value of $\mathcal{O}$ and thus costs p 1 RMR. A The $F\&A$ objects we want also support reading, denoted $x = \mathcal{O}$. The cost of reading a $F\&A$ object is charged to the calling process p like the read of a register: When p reads $\mathcal{O}$, if the value hasn't been written to since the last time it read, then p experiences a cache-miss and is charged 1 RMR. If p reads $\mathcal{O}$ but no process has written to $\mathcal{O}$ since p's last read of $\mathcal{O}$, then p isn't charged an RMR.

Compare-and-Swap(abbreviated CAS): An $n-$process CAS object $\mathcal{O}$ allows any of n processes to call $b = CAS(\mathcal{O}, x, y)$ or $z = \mathcal{O}$. If the value of $\mathcal{O}$ and x are equal, then y is stored in $\mathcal{O}$ and $true$ is returned. Else, $false$ is returned. $z = \mathcal{O}$ reads the value of $\mathcal{O}$ and stores it in the local variable z. Any CAS operation on an object $\mathcal{O}$ by process p, whether it succeeds or fails, costs p 1 RMR. A read of $\mathcal{O}$ by process p costs p 1 RMR iff a CAS of $\mathcal{O}$ has happened since p's last read of $\mathcal{O}$. Else, a read costs process p 0 RMR's.

All three of these objects were used in the algorithm in [1], and are commonly studied objects in the field of distributed computing. It is reasonable to assume that atomic registers are supported on most modern architectures. In [4], it is stated that CAS is supported on many modern architectures, such as multiprocessors based on UltraSparc and Itanium processors.

The state/configuration of the system at any point in the execution of an algorithm can be described by the values of shared variables, local variables, and the program counter (the line number in the algorithm) of each process. We will use the same conventions for steps, runs, and reachability as [1], as quoted from this below:

> "A step is a triple (C, p, C') such that C and C' are configurations, p is a process name, and the execution of a program statement by p in C results in C' . We say p took the step, and C and C' are the start and end configurations of the step respectively. A run from a configuration C is a (finite or infinite) sequence of steps

$(C, p_0, C_1), (C_1, p_1, C_2), (C_2, p_2, C_3)$ where the first configuration is C and the end configuration of each step is the start configuration of the next step. A run refers to a run from the initial configuration. A configuration C is reachable if either C is the initial configuration or there is a finite run such that C is the end configuration of the last step of."

3. Properties

At any configuration, a process can be of exactly one of the following 10 types: normal reader/writer, upgrading reader/writer, downgradingreader/writer, upgraded reader/writer, downgraded reader/writer.

A normal process is one that has neither upgraded nor downgraded in its most recent attempt. An upgrading/downgrading process is one that is currently upgrading/downgrading. An upgraded/downgraded process is one that has upgraded or downgraded respectively in its most recent attempt.

A process is a reader/writer based on what entry section it executed in its most recent attempt.

An *attempt* is a particular execution of the Try and Exit sections by a process. A process *crashes* in an run σ if it takes only finitely many steps in the run. An attempt by r doorway precedes an attempt by r' if r completes the doorway in its attempt before r' begins its doorway.

Mutual Exclusion: If a normal writer or upgraded process is in the CS, then no other process is in the CS

Bounded Exit: There is a b such that in every run, every process completes the exit section in at most b of its steps

First-Come-First-Served (FCFS) among writers: If w and w' are any two writer attempts in a run and w doorway precedes w', then w' does not enter the CS before w

First-in-First-Enabled (FIFE) among readers: Let r and r' be any two reader attempts in a run such that r doorway precedes r'. If r' enters the CS before r, then r is enabled to enter the CS at the time r' enters the CS

Concurrent Entering: There is an integer b such that, if σ is any run from a reachable configuration such that all writers are in the Remainder section in every configuration in σ and there are no upgraded or upgrading readers in the CS or exit section, then every read attempt in σ executes at most b steps of the Try section before entering the CS.

Starvation Freedom: If no process crashes in an infinite run, then all attempts complete in that run

Upgradeability: If throughout an upgrade all readers are in the remainder section, then the upgrade will succeed.

Bounded Upgrade: There is a b such that in every run, every upgrading process completes the upgrade procedure in at most b of its steps

Bounded Downgrade: There is a b such that in every run, every downgrading process completes the downgrade procedure in at most b of its steps

Constant RMR: There is a b such that every attempt uses at most b RMR's, not including the upgrade and downgrade procedures.

These properties are derived from properties stated in [1], modified to account for upgrading and downgrading.

```
Shared Variables
X ∈ {0,1} × ℕ        init (0,0)
Y ∈ ℕ × ℕ            init (0,0)
U ∈ {0,1,2}          init 0
V ∈ {true, false}    init true
W ∈ {0,1}            init 0
Flag ∈ {true, false} init false

Writer-Entry
1  (−, a) = F&A(X, (1,0))
2  (−, b) = F&A(Y, (a,0))
   if a ≠ b
3    wait till Flag = true
4    Flag = false
5  F&A(Y, (−a,0))

Writer-Exit
6  W = W̄

Reader-Entry
7    (c, −) = F&A(X, (0,1))
7.1  CAS(U, 1, 0)
7.2  if (U = 2)
7.3    wait till V = true
8    wait till W = c

Reader-Exit
9   (d, e) = F&A(Y, (0,1))
    if (d = e + 1)
10    Flag = true

Upgraded-Exit
10.1 V = true
10.2 U = 0
10.3 (d, e) = F&A(Y, (0,1))
     if (d = e + 1)
10.4   Flag = true

Upgrade
11.1 (−, a) = Y
11.2 (−, b) = X
     if b − a = 1
11.3   if CAS(U, 0, 1)
11.4     (−, c) = X
         if c = b
11.5       V = false
11.6       return CAS(U, 1, 2)
11.7     else U = 0
     return false

Writer-Downgrade
12.1 F&A(X, (0,1))
12.2 W = W̄

Upgraded-Downgrade
13.1 V = true
13.2 U = 0
```

FIGURE 4.1. Single-Writer Algorithm

4. Single-Writer Algorithm and Intuition

The original algorithm, consisting of all non-numbered lines (i.e. lines 1-10 not including 7.1-7.3) was developed by Jayanti and Liu. This algorithm is a single-writer, multi-reader starvation-free and writer-priority reader-writer exclusion algorithm. Note: it is possible to obtain starvation-freedom and writer-priority since only one writer is allowed. With more than one writer and writer priority, readers could starve as described earlier.

4.1. **Original Algorithm, Without Upgrade/Downgrade.** The intuition behind the algorithm in is as follows:

$X = (a, b)$, where a is the number of times the writer has completed the doorway modulo 2, and b is the number of times readers passed through the doorway.

$Y = (a, b)$, where a is the number of times readers have completed the doorway before the current writer has entered, 0 if the writer is in the doorway or remainder section. b is the number of readers that have begun exiting.

$W \in \{0, 1\}$ the number of times the writer has exited modulo 2

$Flag \in \{true, false\}$ whether the writer is enabled to enter the CS.

Line-by-line commentary:

1. The writer is executing the doorway, just line 1, so it increases $X[0]$ and saves the number of readers, $X[1]$, for future reference. Note: any readers after this point will have executed their doorway after the writer, so the number of readers that doorway precede the writer is exactly the number of readers saved from $X[1]$.

2. The writer records the number of reader attempts that doorway precede it in $Y[0]$ by incrementing $Y[0]$ by that amount.

3. If there are readers that have completed the doorway but not begun exiting, then the writer waits on the readers

4. The writer waits until the last reader to enter before it wakes the writer up.

5. The writer resets $Y[0]$ to 0

6. The writer increments mod 2 the number of times it has exited, enabling readers waiting on W to enter the CS.

7. The reader increments the entry count in $X[1]$ and saves $X[0]$, which indicates whether which attempt a writer is in.

8. The reader waits until the writer is in the remainder section, ie when the number of times the writer has exited is the number of times the writer has entered mod 2.

4.2. **With Upgrade/Downgrade.** $U \in \{0, 1, 2\}$, where 0 indicates no process is attempting to upgrade, 1 indicates a process is attempting to upgrade, and 2 indicates a process will succeed in upgrading

$V \in \{true, false\}$ *true* readers wait on V instead of U when a process is upgrading. This separation of U and V is to decrease RMR's

7.1 If an upgrading is happening that didn't record my increment of X, stop it by setting U back to 0

7.2 If an upgrade succeeded, wait on V, which the upgraded process will change to wake me up.

7.3 Wait on V

10.1 I've upgraded, now let waiting readers go by setting $V = true$

10.2 Allow future upgrades. If U stays 2, then no process will be able to successfully CAS U.

10.3 Wake up the writer by executing the reader exit section.

11.1 Get the number of readers that have entered. This is done after getting the number of exited readers to ensure, if $\#entered - \#exited = 1$ at the moment $\#entered$ is checked, readers are only in lines 7,10, or 10.4. Since I am upgrading, I am registered as a reader in X (even if I started as a writer), so I will account for the 1 in the difference between entered and exited readers.

11.2 There are no readers but me except in lines 7,10, or 10.4. This check prevents multiple readers from upgrading simultaneously, as one would detect the other's presence and fail immediately. The reason to avoid this: let's say lines 11.1 and 11.2 replaced line 11.4, and the if statement in line 11.2 was removed. Let's say only a single reader is outside the remainder section, and it executes until reaching the $CAS(U, 1, 2)$ line. Just before it executes this line, another reader

executes through line 7.1, setting $0 \to U$, all the way to $CAS(U, 0, 1)$ in upgrade. Now U is 1, and the original reader will succeed in $CAS(U, 1, 2)$, giving and an $(upgraded, reader)$ and $(normal, reader)$ in the CS simultaneously.

11.3 Since only one upgrading process is at this point, setting U to 1 provides a way to check whether a reader has executed line 7.1.

11.4 Check that no more readers have entered. At this line, if $c = b$ then I am the only reader not in lines 7,10, or 10.4.

11.5 Setting V here won't block any readers, since no readers are at line 7.3

11.6 If an upgrade gets to this line, when it executed

11.7 I failed to upgrade, so set U back to 0 in case it is still 1

12.1 Pretend I'm an entering reader. I, now a writer, am going to "become" a reader

12.2 Exit as a writer, now I'm fully a reader

13.1 Setting $V = true$ frees all processes waiting on upgraded processes.

13.2 Setting $U = 2$ allows more processes to upgrade.

5. Invariants

$\mathbf{Inv_1} := PC_w = 1 \Rightarrow$
- 1 $Flag = false$
- 2 $X.1 = W$
- 3 $Y.1 = 0$
- 4 $\forall r, (PC_r \in \{7.1 - 8\} \Rightarrow W = r.c)$
- 5 $\forall r, PC_r \notin \{10, 10.4\}$

$\mathbf{Inv_2} := PC_w = 2 \Rightarrow$
- 1 $Flag = false$
- 2 $X.1 \neq W$
- 3 $Y.1 = 0$
- 4 $[\#r : (PC_r \in \{7.1 - 8\} \wedge W = r.c) \vee$
  $PC_r \in \{9, 10.1 - 10.3, 11.1 - 12.2\}] = w.a - Y.2$
- 5 $\forall r, PC_r \notin \{10, 10.4\}$

$\mathbf{Inv_3} := PC_w = 3 \Rightarrow$

$Flag = true \Rightarrow$
- 1 $X.1 \neq W$
- 2 $Y.1 = Y.2 = w.a$
- 3 $\forall r, ((PC_r \in \{7.1 - 8\} \Rightarrow r.c \neq W) \wedge PC_r \notin \{9 - 12.2\})$

$Flag = false \Rightarrow$
- 4 $X.1 \neq W$
- 5 $Y.1 = w.a$
- 6 $[\#r : (PC_r \in \{7.1 - 8\} \Rightarrow r.c = W) \vee$
  $PC_r \in \{9, 10.1 - 10.3, 11.1 - 12.2\}] = Y.1 - Y.2$
- 7 $[\#r : (PC_r \in \{10, 10.4\})] = \begin{cases} 1 & \texttt{if } Y.2 = Y.1 \\ 0 & \texttt{if } Y.2 \neq Y.1 \end{cases}$

$\mathbf{Inv_4} := PC_w = 4 \Rightarrow$
- 1 $Flag = true$
- 2 $X.1 \neq W$

3 $Y.1 = Y.2 = w.a$
4 $\forall r, ((PC_r \in \{7.1 - 8\} \Rightarrow r.c \neq W) \wedge PC_r \notin \{9 - 12.2\})$

$\mathbf{Inv_5}$ $:= PC_w = 5 \Rightarrow$
1 $Flag = false$
2 $X.1 \neq W$
3 $Y.1 = Y.2 = w.a$
4 $\forall r, ((PC_r \in \{7.1 - 8\} \Rightarrow r.c \neq W) \wedge PC_r \notin \{9 - 12.2\})$

$\mathbf{Inv_6}$ $:= PC_w = 6 \Rightarrow$
1 $Flag = false$
2 $X.1 \neq W$
3 $Y.1 = 0$
4 $Y.2 = w.a$
5 $\forall r, ((PC_r \in \{7.1 - 8\} \Rightarrow r.c \neq W) \wedge PC_r \notin \{9 - 12.2\})$

$\mathbf{Inv_{1-6}}$ $:= Inv_1 \vee ... \vee Inv_6$

$\mathbf{Inv_G}$:= `//global invariant`
1 $[\#r : PC_r \in \{7.1 - 9, 10.1 - 10.3, 11.1 - 12.2\}] = X.2 - Y.2$
2 $X.1, X.2, Y.1, Y.2 \geq 0$
3 $[\#r : PC_r \in \{10.1 - 10.3, 11.3 - 11.6, 12.1, 12.2\}] \leq 1$
4 $U = 2 \iff \exists r, PC_r \in \{10.1, 10.2, 12.1, 12.2\}$
5 $\exists r, PC_r \in \{10.1, 11.6, 12.1\} \Rightarrow V = false$
6 $\forall r, (PC_r \in \{11.5, 11.6\} \wedge U = 1) \Rightarrow$
$(\forall q \neq r, PC_q \notin \{7.2 - 9, 10.1 - 10.3, 11.1 - 12.2\})$
7 $\forall r, PC_r \in \{10.1, 12.1\} \Rightarrow$
$(\forall q \neq r, PC_q \notin \{8, 9, 10.1 - 10.3, 11.1 - 12.2\})$
8 $(\forall r, PC_r \notin \{10.1, 12.1\}) \Rightarrow$
$(V = true \vee (U \neq 2 \wedge \forall q, PC_q \neq 7.3))$
9 $Y.2 \geq r.a$
10 $\exists r, PC_r \in \{11.3 - 11.6\} \Rightarrow \forall q, PC_q \neq 7.3$
11 $\forall r, PC_r \notin \{11.4 - 11.7\} \Rightarrow U \neq 1$

6. Proof of Properties From Invariants

Lemma 1. *Informally: For some reader r, once $W = r.c$ it stays that way until leaving the try section. Formally: Let r be a reader in a reachable configuration C such that $PC_r \in \{7.1 - 8\}$ and $W = r.c$. If after a single step (C, q, C'), $PC_r \in \{7.1 - 8\}$ in C', then $W = r.c$ in C'.*

Since $r.c$ is local to r, $r.c$ is not modified in lines $7.1 - 8$, and r stays on lines $7.1 - 8$, the step doesn't change $r.c$. It is thus sufficient to show W hasn't changed from C to C'.

For $W = rc$ to hold in C, the writer can't be at line 6, since $Inv_6.5$ would imply $r.c \neq W$. Notice that line 6 is the only line where W is changed. Thus, the step won't change W.

Mutual Exclusion. If a normal writer is in the CS, then the writer is at line 6 and by $Inv_6.5$, no reader is in the CS.

If an upgraded process p is in the CS, then $PC_p \in \{10.1, 12.2\}$ so directly by $Inv_G.3$, no other process is in the CS.

First-Come-First-Served (FCFS) among writers. Ensured by the FCFS lock

First-in-First-Enabled (FIFE) among readers. Since r' enters the CS, at some point it is at line 8 and $W = r'.c$. If $r.c = r'.c$, then when $W = r'.c$ $W = r.c$, so by lemma 1 r will not be blocked at line 8. If $r.c \neq r'.c$, then after r executes line 7 and before r' executes line 7, $X.1$ is must have been increased. Since $X.1$ is only changed at line 1, the writer must have been at line 1 and executed 1 between the time r executes 7 and r' executes 7. This means $Inv_4.4$, $W = r.c$, so r will not be blocked at 8 after the point when the writer is at line 1.

If r is blocked or will be blocked in bounded number of its own steps, $V = false \wedge (U = 2 \vee \exists q, PC_q = 7.3)$. By $Inv_G.8$, this means $\exists s, PC_s \in \{10.1, 12.1\}$. By $Inv_G.7$, this means r' is not in the CS while r is blocked. Thus at any point while r' is in the CS, r is not blocked. Even when r' leaves the CS, r stays unblocked since V will not be able to change back to $false$, and U will not be able to change back to 2 because no process will be able to execute line 11.4 successfully. This follows directly from $Inv_G.1$, the fact that r is on one of lines $7.1 - 8$, and $r.a \leq Y.2$ since Y only increases.

Concurrent Entering. Concurrent entering assumes all writers are in remainder and there are no upgraded processes in the CS or exit section. Since the writer is at line 1, by $Inv.1$ a reader will never be blocked at line 8. Since there are no upgraded reader processes in the CS or exit section, there are no processes at lines 12.1 or 10.1, so by $Inv_G.8$ a reader won't wait at line 7.3.

Starvation Freedom. A writer can't starve: Assume for writer w, $PC_w = 3$ and $Flag = false$. Since $Y.1, Y.2 \geq 0$ by $Inv_G.2$, either $Y.1 - Y.2 > 0$ or $Y.1 - Y.2 = 0$. If $Y.1 - Y.2 > 0$, then by $Inv_3.6$ there are $Y.1 - Y.2$ readers ready to execute one of lines 9 or 10.3 in a bounded number of their own steps, unless they are blocked at 7.3. A reader r that is blocked at 7.3 will only be blocked until the upgraded process finishes exiting, and at this point by the argument for FIFE, r will not be blocked at 7.3 in the current attempt. The last to execute 9 or 10.3 will do so when $Y.1 - Y.2 = 1$, and will enter line 10 and set $Flag = true$.

Else, $Y.1 = Y.2$, so by $Inv_3.7$ some process will be poised to set $Flag = true$.

A reader can't starve: Assume there is a reader r such that $PC_r = 8$ and $W \neq r.c$. Since the writer doesn't starve, it will at some point end up at line 6. When the writer executes line 6, it will then be at line 1, and $W = r.c$. By Lemma 1 W will stay equal to $r.c$ until r enters the CS.

Assume reader is at line 7.3 and $V = false$. The contrapositive of $Inv_G.4$ says $(V = false \wedge (U = 2 \vee \exists q, PC_q = 7.3)) \Rightarrow (\exists r, PC_r \in \{10.1, 12.1\})$. By this, there is a process t ready to execute one of lines 10.1 or 12.1 and set $V = true$.

Upgradeability. We need to show that if all readers are in the remainder section, an upgrade will succeed. Upgrade can fail at lines 11.2, 11.3, 11.4, and 11.6. Upgrade won't fail at line 11.2 since there have been no other readers in lines $\{7.1 - 9, 10.1 - 10.3, 11.1 - 12.2\}$ while lines 11.1 and 11.2 were executed, so $b - a = 1$ by $Inv_G.1$.

Upgrade won't fail at line 11.3 since by $Inv_G.11$ $U \neq 1$.

Upgrade won't fail at line 4 since no readers have entered, and $X.2$ hasn't changed.

Upgrade won't fail at line 11.6 since there have been no other readers to execute lines 11.7 and 7.1, so $U = 1$.

Bounded Upgrade/Downgrade/Exit. Look at the algorithm: for the upgrade/downgrade/exit procedures, there are finitely many non-looping steps and all of the complete immediately

Constant RMR. Every line in the Writer-Entry, Reader-Entry, Writer-Exit, Reader-Exit, and Upgraded-Exit section except 3, 7.3, and 8 complete using at most 1 RMR. We will show at each of these lines, a process incurs constant RMR's before proceding.

A writer waiting at 3 will take an RMR hit iff $Flag$ is written (even if it isn't changed, recall we're not assuming smart cache). Flag can only be written at lines 4, 10, and 10.4. Line 4 won't be executed since the writer is at 3, and there is only one writer. Once one of lines 10 or 10.4 is executed, $Flag = true$, and since $Flag$ is only set to $false$ at line 4, $Flag$ stays true. By $Inv_3.3$ from this point on non processes can be at line 10 or 10.4, so $Flag$ can't be written again. Thus a waiting writer at 3 will be charged at most one RMR.

If there is a reader at 7.3, then by the contrapositive of $Inv_G.10$ there are no processes at lines $11.3 - 11.6$. The only processes that can set V are at 10.1 and 12.1, and by $Inv_G.3$ there is at most 1. After this process leaves, since no processes can be at $11.3 - 11.6$ while the reader is at 7.3, no processes execute lines 10.1 or 12.1 before the reader leaves line 7.3.

If there is a reader at line 8, by the argument in Lemma 1 below W is changed only once, so only one RMR is charged.

7. Proof of Invariants

Inv_{1-6} holds in the initial configuration: $Inv_2, ...Inv_6$ hold trivially since $PC_w = 1$. Inv_1 also holds since $Flag' = false$, $X.1 = 0 = W$, $Y.1 = 0$, and there are no readers at lines $7.1 - 10$.

Inv_G holds in the initial configuration. As mentioned in the statement of the algorithm, initially $X = (0, 0)$, $Y = (0, 0)$, $U = 0$, $V = true$, $W = 0$, and $Flag = false$. $X.1 = X.2 = Y.1 = Y.2 = 0$, and all processes are in the remainder section (ie $PC_w = 1$ and $\forall r, PC_r = 7$). $Inv_G.1$ holds: $X.2 - Y.2 = 0$ and there are no processes at lines $7.1 - 9, 10.1 - 10.3, 11.1 - 12.2$. $Inv_G.2$ holds: $X.1, X.2, Y.1, Y.2 \geq 0$. $Inv_G.3$ holds: there are also no processes at $10.1 - 10.3, 11.3 - 11.7, 12.1, 12.2$. $Inv_G.4$ holds: $U \neq 2$ and there are no processes at 10.1...12.2. $Inv_G.5$ holds: There are no processes at $10.1, 11.7, 12.1$. $Inv_G.6$ holds: There are no processes at lines $11.5, 11.6$. $Inv_G.7$ holds: There are no processes at $10.1, 12.1$. $Inv_G.8$ holds: $V = true$ makes $Inv_G.8$ $(something) \Rightarrow true$, or $true$.

> **Note:** We will discuss the various scenarios where some process is at a particular line and takes a step. When discussing a variable, let's say X, in such scenarios, we will use the convention: X to talk about the variable and the value of the variable before the step, and X' to represent the value of the variable after the step.

Assume Inv_{1-6} holds when $PC_w = 1$, and a reader takes a step. Then Inv_{1-6} still holds: Inv_1 still holds: no reader can change $Flag$ (since by Inv_1, no reader can be at lines $10, 10.4$), $X.1$, $Y.1$, or W. Since $W' = W$, and a reader can only change its $r.c$ or enter lines $7.1 - 8$ by executing line 7, a reader step can only violate $\forall r, PC'_r \in \{7.1 - 8\} \Rightarrow r.c' = W'$ by executing line 7. When a reader executes line 7, by Inv_1 $X.1 = W$, so when that reader enters line 7.1 $r.c' = W'$. A reader will never enter lines 10 or 10.4 since at lines 9 and 10.3, by Inv_1 $Y.1 = 0$, by Inv_G $Y.2 \geq 0$, so $Y.1 \neq Y.2 + 1$.

$Inv_2, ...Inv_6$ hold trivially since $PC'_w = 1$.

Assume Inv_{1-6} holds when $PC_w = 1$, and a writer takes a step. Then Inv_{1-6} still holds: Inv_2 holds: $Flag, X.2, Y.1, Y.2, W$ don't change, so $Flag' = false$ and $Y.1' = 0$. Since only the writer is taking a step at line 1, and Inv_1 held before the step, no readers will enter lines 10 or 10.4. $w.a' = X.2'$, so $w.a' - Y.2' = X.2 - Y.2 = [\#r : PC'_r \in \{7.1 - 9, 10.1 - 10.3, 11.1 - 12.2\}]$ by the global invariant, since $\forall r, PC_r = PC'_r$. Since $\forall r, r.c' = r.c$, $W' = W$, and $PC'_r = PC_r$, $\forall r, PC'_r \in \{7.1 - 8\} \Rightarrow W' = r.c'$, so $[\#r : PC'_r \in \{7.1 - 9, 10.1 - 10.3, 11.1 - 12.2\}] = [\#r : (PC'_r \in \{7.1 - 8\} \wedge W' = r.c') \vee PC'_r \in \{9, 10.1 - 10.3, 11.1 - 12.2\}]$, and thus $[\#r : (PC'_r \in \{7.1 - 8\} \wedge W' = r.c') \vee PC'_r \in \{9, 10.1 - 10.3, 11.1 - 12.2\}] = w.a' - Y.2'$. Since $X.1 = W$ Inv_1, and $X.1' = X.1 + 1$, $X.1' \neq W'$.

$Inv_1, Inv_3, ...Inv_6$ hold trivially since $PC'_w = 2$.

Assume Inv_{1-6} holds when $PC_w = 2$, and a reader takes a step. Then Inv_{1-6} still holds: Inv_2 still holds: no reader can change $Flag$ (since no reader can be in lines 10 or 10.4 by Inv_2), $X.1$, $Y.1$, or W. Since $X.1 \neq W$, the number of readers at lines $7.1 - 8$ such that $W = r.c$ will not change from a reader executing line 7. Thus, the only way for $[\#r : (PC_r \in \{7.1 - 8\} \wedge W = r.c) \vee PC_r \in \{9, 10.1 - 10.3, 11.1 - 12.2\}]$ to change is for a reader to execute one of lines 9 or 10.3. No reader will change $w.a$, and a reader will only change $Y.2$ on lines 9 or 10.3, so $w.a - Y.2$ will only change by a reader executing lines 9 or 10.3. If a reader executes line 9 or 10.3, $[\#r : (PC_r \in \{7.1 - 8\} \wedge W = r.c) \vee PC_r \in \{9, 10.1 - 10.3, 11.1 - 12.2\}]$ will decrease by 1 and $w.a - Y.2$ will decrease by 1, maintaining the invariant. A reader will not enter lines 10 or 10.4 since at lines 9 and 10.3, by Inv_2 $Y.1 = 0$, by Inv_G $Y.2 \geq 0$, so $Y.1 \neq Y.2 + 1$.

$Inv_1, Inv_3, ...Inv_6$ hold trivially since $PC'_w = 2$.

Assume Inv_{1-6} holds when $PC_w = 2$, and a writer takes a step. Then Inv_{1-6} still holds: Inv_3 holds: Only $Y.1$ has changed (increased by $w.a$), so by Inv_2 $Flag' = false$, $X.1' \neq W'$, and $Y.1' = w.a' + 0 = w.a'$ by Inv_2. Thus $Y.1' - Y.2' = w.a' - Y.2' = [\#r : (PC'_r \in \{7.1 - 8\} \wedge W' = r.c') \vee PC'_r \in \{9, 10.1 - 10.3, 11.1 - 12.2\}]$ since only the writer took a step. When the writer executes line 2 and ends up on line 3, the if statement on line 2 succeeded so $w.a \neq Y.2$ and $w.a' \neq Y.2'$. Thus $Y.1' = w.a' \neq Y.2'$. We know from Inv_2 that $[\#r : PC_r \in \{10, 10.4\}] = 0$, so $Inv_3.7$ holds.

Inv_5 holds: Only $Y.1$ has changed (increased by $w.a$), so by Inv_2 $Flag' = false$, $X.1' \neq W'$, and $Y.1' = w.a' + 0 = w.a'$ by Inv_2. When the writer executes line 2 and ends up on line 5, the if statement on line 2 succeeded so $w.a = Y.2$ and $w.a' = Y.2'$. Because $w.a = Y.2$, by $Inv_2.4$ $[\#r : (PC'_r \in \{7.1 - 8\} \wedge W' = r.c') \vee PC'_r \in \{9, 10.1 - 10.3, 11.1 - 12.2\}] = 0$, which combined with $Inv_2.5$ implies $Inv_5.4$.

$Inv_1, Inv_2, Inv_4, Inv_5, Inv_6$ hold trivially since $PC'_w = 3$.

Assume Inv_{1-6} holds when $PC_w = 3$, and a writer takes a step, staying at 3. Then Inv_{1-6} still holds: No local or shared variables have changed, and no program counters have changed, so Inv_{1-6} trivially holds.

Assume Inv_{1-6} holds when $PC_w = 3$, and a reader takes a step. Then Inv_{1-6} still holds: Inv_3 still holds: Readers don't change $X.1$ and W, so $X.1' \neq W'$. Readers don't change $Y.1$ either, so $Y.1' = w.a'$.

- If $Flag = true$, since $Inv_3.3$ says there are no readers at lines 10 or 10.4 before, $Flag' = true$. The only way line $Inv_3.3$ could conceivably change from the reader step is if a reader executed line 7 or line 8. If a reader r executes line 7, since $X.1 \neq W$, $r.c' \neq W'$ and $Inv_3.3$ still holds. If a reader executes line 8, we know $r.c \neq W$, so the reader stays on 8 and $Inv_3.3$ isn't affected. No reader was at line 9 or 10.3 before by $Inv_3.3$, so $Inv_3.2$ means $Y.2$ doesn't change and implies $Inv_3.2$ after.
- If $Flag = false$, then *the* reader taking a step is at one of lines 10 or 10.4, or it is not. If the reader is at one of these lines, then $Flag' = true$ and this is the only change that happens, so by Inv_3 $X.1' = W', Y.1' = w.a'$, and by $Inv_3.7$ $Y.2' = Y.1'$. Thus $Y.1' - Y.2' = 0$, so by $Inv_3.6$ before the step combined with $Inv_3.7$ before and the fact that the reader is leaving line 10/10.4 implies $Inv_3.3$ after the step.

  Else, the reader is taking a step at some line that is not 10 or 10.4, so $Flag' = false$. No reader can change $X.1, Y.1$ or W so $X' \neq W'$ and $Y.1' = w.a'$. A reader can only conceivably violate $Inv_3.6$ if it executes one of lines 7, 8, or 9/10.3. A reader r that executes line 7 will have $r.c' \neq W'$ since by $Inv_3.4$ $X.1 \neq W$, not changing either side of $Inv_3.6$. If r executes lin 8, then $PC'_r \in \{9, 11.1\}$, and $Y.1/Y.2$ aren't changed, so neither side of $Inv_3.6$ changes and $Inv_3.6$ holds. If r executes 9/10.3, it increases $Y.2$ by 1 and is no longer counted in the left side of $Inv_3.6$, so $Inv_3.6$ holds. If some reader is at lines 10 or 10.4, then by $Inv_3.7$ $Y.2 = Y.1$, so $Y.1 - Y.2 = 0$ and there are no processes at lines 9 or 10.3 before. This combined with the fact that the processes at lines 10 and 10.4 don't execute imply $Inv_3.7$ holds after the step. If no readers are at lines 10 or 10.4, then $Y.1 \neq Y.2$ so no reader will execute lines 9/10.3 and end up at 10/10.4, meaning $Inv_3.7$ holds after the step.

Assume Inv_{1-6} holds when $PC_w = 3$, and a writer takes a step to line 4. Then Inv_{1-6} still holds: Inv_4 holds: This step will only happen when $Flag = true$, so $Flag' = true$, $X.1' \neq W'$, $Y.1' = Y.2' = w.a'$, and $\forall r((PC'_r \in \{7.1 - 8\} \Rightarrow r.c' \neq W') \wedge PC'_r \notin \{9 - 12.2\})$.

$Inv_1, ...Inv_3, Inv_5, Inv_6$ hold trivially since $PC'_w = 4$.

Assume Inv_{1-6} holds when $PC_w = 4$, and a reader takes a step. Then Inv_{1-6} still holds: Inv_4 still holds: There are no readers at lines $10, 10.4$ by $Inv_4.4$, so $Flag' = true$. No reader changes $X.1$, $Y.1$, or W, so $X.1' \neq W'$ and $Y.1' = w.a'$. No reader is at lines 9 or 10.3 so $Y.2$ doesn't change either, and thus $Y.2' = w.a'$. Since $\forall r, PC'_r \in \{7.1 - 8\} \Rightarrow r.c \neq W$, no readers will enter lines 9/11.1, so there are still no readers on lines $9 - 12.2$. Since $X.1 \neq W$, a reader that executes line 7 will have $r.c' \neq W'$ when at lines $7.1 - 8$.

$Inv_1, Inv_2, Inv_3, Inv_5, Inv_6$ hold trivially since $PC'_w = 4$.

Assume Inv_{1-6} holds when $PC_w = 4$, and a writer takes a step. Then Inv_{1-6} still holds: Line 4 just sets $Flag = false$, and Inv_4 and Inv_5 are identical except for $Flag$ being inverted.

Assume Inv_{1-6} holds when $PC_w = 5$, and a reader takes a step. Then Inv_{1-6} still holds: Identical argument to $PC_w = 4$ and a reader taking a step, except $Flag$ stays at $false$ this time.

Assume Inv_{1-6} holds when $PC_w = 5$, and a writer takes a step. Then Inv_{1-6} still holds: Inv_6 holds: Only $Y.1$ has changed, so $Flag' = false$, $X.1' \neq W'$, and $Y.2' = w.a'$. We see $\forall r, r.c$ hasn't changed since the writer executed a step, not a reader, and W hasn't changed, so $\forall r((PC_r \in \{7.1 - 8\} \Rightarrow r.c \neq W) \wedge PC_r \notin \{9 - 12.2\})$ holds. Since $Y.1 = w.a$, and $Y.1$ is decreased by $w.a$, $Y.1' = 0$

$Inv_1, ..., Inv_5$ hold trivially

Assume Inv_{1-6} holds when $PC_w = 6$, and a reader takes a step. Then Inv_{1-6} still holds: Inv_6 still holds: By Inv_6 no reader is on lines 10/10.4, so $Flag$ doesn't change and $Flag' = false$. Since a reader takes a step $X.1$, W, and $Y.1$ don't change either, so $X.1' \neq W'$, and $Y.1' = 0$. Since $X.1 \neq W$, any reader that takes a step from line 7 will have $r.c \neq W$ when at one of lines $7.1 - 8$. By $Inv_6.5$, a reader r that executes line 8 won't end up at lines 9 or 11.1 since $r.c \neq W$, so $PC_r \notin \{9 - 12.2\}$. Since no reader executes lines 9 or 10.3, and no writer is executing, neither $w.a$ or $Y.2$ change, so $Y.2' = w.a'$.

Assume Inv_{1-6} holds when $PC_w = 6$, and a writer takes a step. Then Inv_{1-6} still holds: Only W has been changed to $\overline{W}$. After the step is taken, $Flag' = false$, $X.1' = W'$ since W was flipped, $Y.1' = 0$, $\forall r((PC'_r \in \{7.1 - 8\} \Rightarrow r.c' \neq W'))$ since only W was flipped, and since $\forall r, PC_r \notin \{9 - 12.2\}$ by $Inv_6.5$, $\{r \mid PC'_r = 10\} = \emptyset$ after the step.

Assume Inv_G and Inv_{1-6} hold and any process takes a step. Then $Inv_G.1$ holds: The only way either side of the equation could change is if a process takes a step at 7, 9, or 10.3. If a process takes a step at 7, the left side increases by 1 and $X.2$ increases by 1. If a process takes a step at 9, the left side decreases and $Y.2$ increases by 1. Likewise for 10.3. Thus $Inv_G.1$ still holds after the step.

Assume Inv_G and Inv_{1-6} hold and any process takes a step. Then $Inv_G.2$ holds: $X.1, X.2, Y.1, Y.2$ only increase except at line 5. At this line Inv_5 says $Y.1 = w.a$, and $Y.1$ is decreased by $w.a$, so $Y.1 \geq 0$ after taking a step.

Assume Inv_G and Inv_{1-6} hold and any process takes a step. Then $Inv_G.3$ holds: The number of readers in $10.1 - 10.3, 11.3 - 11.7, 12.1, 12.2$ can only increase if a reader executes line 11.2 successfully. Since for any reader at line 11.2, $r.a \leq Y$, the reader will only successfully complete line 11.2 if there are no other readers in the lines given in $Inv_G.1$. The lines in $Inv_G.1$ are a superset of the lines given here, so no 2 readers will enter the lines given here.

Assume Inv_G and Inv_{1-6} hold and any process takes a step. Then $Inv_G.4$ holds: Either a process starts outside $10.1, 10.2, 12.1, 12.2$ and enters lines $10.1, 10.2, 12.1, 12.2$, it stays at $10.1, 10.2, 12.1, 12.2$, or it starts at $10.1, 10.2, 12.1, 12.2$ and leaves them.

A process can only enter these lines by executing line 11.6 and succeeding. This can only happen if $U = 1$, so $U \neq 2$ and there are no processes at lines $10.1, 10.2, 12.1, 12.2$ before the step. After the step, $U = 2$ and there is a process at one of lines $10.1, 12.1$.

A process can only stay inside $10.1, 10.2, 12.1, 12.2$ by executing $10.1, 12.1$. This doesn't change U, so the invariant still holds.

A process can only leave $10.1, 10.2, 12.1, 12.2$ by executing lines $10.2, 12.2$. By $Inv_G.3$, the process is the only one at lines $10.1, 10.2, 12.1, 12.2$. Both lines $10.2, 12.2$ set $U = 0$. Thus after the step, $U \neq 2$ and there are no processes at $10.1, 10.2, 12.1, 12.2$.

Assume Inv_G and Inv_{1-6} hold and any process takes a step. Then $Inv_G.5$ holds: The only way this invariant could be violated is if a process steps into these lines. A process can only step into 10.1 from 11.6. By $Inv_G.5$ and the fact that V isn't change at line 11.6, $V' = false$. A process can only step into 11.6 from 11.5, which sets $V = false$. A process can only step into 12.1 from 11.6, so by $Inv_G.5$ and the fact that V isn't change at line 11.6, $V' = false$

Assume Inv_G and Inv_{1-6} hold and any process takes a step. Then $Inv_G.6$ holds: For each reader r, either the reader will enter lines $11.5, 11.6$, stay inside them, or leave them. If a reader enters from line 11.4, the only way it can enter, $r.c = X.2$. Since by $Inv_G.9$ $Y.2 \geq r.a$, and $r.b$ hasn't changed since line 11.2 (it's local!), then $Y.2 \geq r.a = r.b - 1 = r.c - 1 = X.2 - 1$, so $X.2 - Y.2 \leq 1$. By $Inv_G.1$ this means no other processes are at lines $7.1 - 9, 10.1 - 10.3, 11.1 - 12.2$. Thus the invariant holds.

If reader r stays inside $11.5, 11.6$ for a step, this means it executes line 11.5 or another reader executes. If r executes 11.5, clearly $Inv_G.6$ isn't affected. Another reader could only conceivably violate the invariant for r in only 2 ways: by changin U to 1 or by stepping into lines $7.2 - 9, 10.1 - 10.3, 11.1 - 12.2$. The only way U could change to 1 is by executing line 11.3, and this can't happen since there can be no reader here, because of $Inv_G.3$ and $PC_r \in \{11.5, 11.6\}$. A reader can only step into $7.2 - 9, 10.1 - 10.3, 11.1 - 12.2$ by executing line 7.1, but as a result of executing this line no matter what $U \neq 1$ after, so the invariant holds.

If a reader r leaves lines $11.5, 11.6$, it can only do so by executing line 11.6. No matter what, after executing line 11.6 $U \neq 1$, so the invariant holds.

Assume Inv_G and Inv_{1-6} hold and any process takes a step. Then $Inv_G.7$ holds: A reader r can only enter 10.1 or 12.1 by executing line 11.6 successfully, and for this to happen U would have to be 1 at line 11.6. By $Inv_G.6$, this would imply there are no other readers at lines $7.2 - 9, 10.1 - 10.3, 11.1 - 12.2$, and the execution of the step isn't going to change that.

If another reader executes while is at lines 10.1 or 12.1, it is only going to violate the invariant by stepping from lines 7.2 or 7.3 to 8. By $Inv_G.4$ $U = 2$, so no reader will step from 7.2 to 8. By $Inv_G.5$, $V = false$, so no reader is going to step from 7.3 to 8. Thus the invariant holds.

Assume Inv_G and Inv_{1-6} hold and any process takes a step. Then $Inv_G.8$ holds: By the inductive hypthesis for $Inv_G.8$, at least one of $\exists r, PC_r \in \{10.1, 12.1\}$, $V =$

true, or $U \neq 2 \wedge \forall q, PC_q \neq 7.3$ must be true. In the first case, the invariant would only be violated by a process taking a step from 10.1 or 12.1. This would make V' true, maintaining the invariant.

In the second case, the invariant could only be violated if a process took a step and set $V = false$. This could only happen at line 11.5. By $Inv_G.3$ this means no other processes are at lines $10.1, 10.2, 12.1, 12.2$, so by $Inv_G.4$ $U \neq 2$. Also, by $Inv_G.10$ there are no processes at line 7.3. Thus $U \neq 2 \wedge \forall q, PC_q \neq 7.3$.

In the last case, the invariant could only be violated if a process set $U = 2$ or a process entered line 7.3. The former could only happen by a process successfully executing line 11.6, which would result in a process at line 10.1 or 12.1. The latter would indicate $U = 2$ before, which was not true.

Assume Inv_G and Inv_{1-6} hold and any process takes a step. Then $Inv_G.9$ holds: $Y.2$ is only increased, and $r.a$ is only changed when it is set to $Y.2$ at line 11.1

Assume Inv_G and Inv_{1-6} hold and any process takes a step. Then $Inv_G.10$ holds: Assume a process enters line 11.3 from 11.2. Then since $Y.2 \geq r.a$ and $r.b - r.a = 1$, it follows that $X.2 - Y.2 = 1$ and there are is no process at line 7.3 by $Inv_G.1$.

If a process is in $11.3-11.7$ and takes a step, $Inv_G.3$ implies there are no processes at lines $10.1, 10.2, 12.1, 12.2$, so $U \neq 2$ and no process enters line 7.3.

Assume Inv_G and Inv_{1-6} hold and any process takes a step. Then $Inv_G.11$ holds: The only place where U can be changed from 0 to 1 is at line 11.3. It will only be changed from 0 to 1 if the CAS succeeds, and in such a case the process will be in line 11.4, making the invariant true.

The only other way the invariant could be violated is by a process leaving $11.4-11.6$. A process can only do this by lines 11.6 and 11.7. A process leaving via 11.6 that succeeds will set $U = 2$, otherwise $U \neq 1$, and U won't be changed so after the CAS $U \neq 1$. A process leaving via 11.7 will set $U = 0$, so $U \neq 1$.

8. Writer Downgrade

Writer downgrade was ignored in the above proofs since it would complicate the already large invariant proof, and it can added in a rigorous manner. We will show that a writer executing the downgrade procedure results in precisely the same configuration as a reader executing until line 8, the reader entering line 9 or 11.1 just after the writer executes line 12.2, and the writer/reader switching PID's. Thus for the single-writer case, the writer becomes a reader in every way. From then on upgrades and downgrades of this "reader" are performed as desired.

9. Multi-Writer

It is possible to obtain a multi-writer Multi-reader starvation-free reader-writer exclusion algorithm from the single-writer version by simply adding a FCFS, Constant RMR Mutex Lock on the incoming writers. When a writer enters, it acquires a lock and records that it entered as a writer. When the writer exits, whether as a reader, an upgraded reader, or as a writer, it releases the lock to allow other writers to enter. Mutual Exclusion isn't violated since the Single-Writer algorithm only encounters one writer at a time. Exit is still bounded since the only addition is the release of the FCFS lock. Since the lock is FCFS, the doorway of the lock functions as the writer's doorway to the algorithm, making the algorithm FCFS among

writers. The remainder of the properties clearly hold, since the inner single-writer algorithm hasn't been changed, and the writer mutex is FCFS and starvation free.

10. Acknowledgments

This wouldn't have been possible without the guidance and help of my thesis advisor, Prasad Jayanti, and the lively discussions with the other members of his group: Vibhor Bhatt, Jack Bowman, Jonathan Choi, Michael Diamond, Zhiyu Liu, and Nancy Zheng. I'm greatly appreciative of my thesis committee members, Robert Drysdale and Sean Smith, for taking the time to review my thesis talk and writ up.

References

[1] Vibhor Bhatt, and Prasad Jayanti. Constant RMR Solutions to Reader Writer Synchronization. *PODC '10*, July 25-28, 2010, Zurich, Switzerland.

[2] James H. Anderson, Yong-Jik Kim, and Ted Herman. Shared-Memory Mutual Exclusion: Major Research Trends Since 1986. Distrib. Computing, 2003.

[3] Vassos Hadzilacos and Robert Danek. Local-Spin Group Mutual Exclusion Algorithms. Proceedings of the 18th International Symposium on Distributed Computing, pg 71–85.

[4] Prasad Jayanti and Srdjan Petrovic. Efficient and Practical Constructions of LL/SC Variables. PODC 2003.

[5] P. J. Courtois, F. Heymans, and D. L. Parnas. Concurrent Control with "Readers" and "Writers". Commun. ACM, 14(10):667–668, 1971.

[6] E. W. Dijkstra. Solution of a Problem in Concurrent Programming Control. Commun. ACM, 8(9):569, 1965.

[7] Leslie Lamport. A New Solution of Dijkstra's Concurrent Programming Problem. Communications of the ACM, August 1974, Volume 7, Number 8.