Roy Portas - 43560846

Q1 a)

The problem with the algorithm is the use of **c**, as its a count used to check if something is writing. When two writers are introduced, both will share the same **c** value and both writers could interrupt each other, causing **c** to be even when the writers are writing. Since **c** is even the algorithm allows the reader to read, thus reading a value that is being written to, which is not correct.

Below is an example of how two writers can break the program:

c	**Writer 1**	**Writer 2**
0	D1 <- get()	
0	D2 <- get()	
1	c <- c+1	
1		D1 <- get()
1		D2 <- get()
2		c <- c+1
2	x1 <- d1	
2	x2 <- d2	

It can be seen on the last two lines that the writer is writing the values when **c** is even, which breaks the algorithm.

To solve this a semaphore can be used to ensure only one writer can increment **c** and update the values at any given time.

Non-blocking reader-writer	
Integer c, x1, x2 <- 0 **Binary semaphore S <- (1, 0)**	
Reader	**Writer**

integer c0, d1, d2
loop forever
  repeat
  repeat
  c0 <- c
  until (c0 mod 2 = 0)
  d1 <- x1
  d2 <- x2
  until (c0 = c)
  use(d1, d2)
	integer c0, d1, d2
	loop forever
	
	  d1 <- get()
	  d2 <- get()
	
	  wait(S)
	  c <- c+1
	  x1 <- d1
	  x2 <- d2
	  c <- c+1
	
	  signal(S)

Using a semaphore ensure that only one writer can enter that section of code. If another writer attempts to writer, it will wait until the semaphore becomes available and immediately takes the lock and proceeds, ensuring that no writer process is ever blocked unnecessarily.

Q1 b)

A monitor can be used to assist with implementing the incrementer, this can be defined as follows:

```
monitor Monitor
    integer writers <- 0
    integer incrementers <- 0

    condition canWrite
    condition canIncrement

    operation writeLock
        if writers != 0 or incrementers != 0
            wait(canWrite)
        writers++

    operation writeUnlock
        if empty(canWrite)
            signal(canIncrement)
        else
            signal(canWrite)
        writers--

    operation incrementerLock
        if writers != 0 or incrementers != 0
            wait(canIncrement)
        incrementers++

    operation incrementerUnlock
        if !empty(canWrite)
            signal(canWrite)
        else
            signal(canIncrement)
        incrementers--
```

This monitor ensures that the writer has higher priority over the incrementer by always signalling the writer if there is a writer waiting. If no writers are waiting it will signal the incrementers to run.

Non-blocking reader-writer-incrementer		
Integer c, x1, x2 <- 0 **Monitor m**		
Reader	**Writer**	**Incrementer**

integer c0, d1, d2
loop forever
  repeat
  repeat
  c0 <- c
  until (c0 mod 2 = 0)
  d1 <- x1
  d2 <- x2
  until (c0 = c)
  use(d1, d2)
	integer c0, d1, d2
	loop forever
	  d1 <- get()
	  d2 <- get()
	  m.writeLock
	  c <- c+1
	  x1 <- d1
	  x2 <- d2
	  c <- c+1
	  m.writeUnlock
		integer c0, d1, d2
		loop forever
		  loop forever
		  repeat
		  c0 <- c
		  until (c0 mod 2 = 0)
		  d1 <- x1
		  d2 <- x2
		
		  m.incrementerLock
		  if c0 == c
		  c <- c + 1
		  x1 = d1 + 1
		  x2 = d2 + 1
		  c <- c + 1
		  m.incrementerUnlock
		  break
		  m.incrementerUnlock