First Readers Writers Problem

Introduction

This algorithm uses semaphores to solve the first readers-writers problem.

Algorithm

```
 1    var     wrt, mutex: semaphore;
 2            readcount: integer  ;
 3    begin
 4            readcount := 0;
 5            wrt := 1;
 6            mutex := 1;
 7    parbegin
 8            repeat                          (* reader process *)
 9                    (* do something *)
10                    down(mutex);
11                    readcount := readcount + 1;
12                    if  readcount = 1 then
13                            down(wrt);
14                    up(mutex);
15                    (* read the file *)
16                    down(mutex);
17                    readcount := readcount - 1;
18                    if  readcount = 0 then
19                            up(wrt);
20                    up(mutex);
21        (* do something else *)
22            until  false;
23            repeat                          (* writer process *)
24                    (* do something *)
25                    down(wrt);
26                    (* write to the file *)
27                    up(wrt);
28                    (* do something else *)
29            until  false;
30    parend  ;
31    end .
```

Comments

lines 1-2 — Here, `readcount` contains the number of processes reading the file, and `mutex` is a semaphore used to provide mutual exclusion when `readcount` is incremented or decremented. The semaphore `wrt` is common to both readers and writers and ensures that when one writer is accessing the file, no other readers or writers may do so.

lines 4-6 — This just initializes all the semaphores. It is the only time anything other than a *down* or an *up* operation may be done to them. As no readers are yet reading the file, `readcount` is initialized to `0`.

line 9 — Since the file is not accessed here, we don't need to put semaphores around this part.

lines 10-15 — Since the value of the shared variable `readcount` is going to be changed, the process must wait until no-one else is accessing it (`down(mutex)`). Since this process will read from the file, `readcount` is incremented by 1; if this is the only reader that will access the file, it waits until any writers have finished (`down(wrt)`). It then indicates other processes may access `readcount`

(*down*(*mutex*)) and proceeds to read from the file.

lines 16-20 Now the reader is done reading the file (for now.) It must update the value of *readcount* to indicate this, so it waits until no-one else is accessing that variable (*down*(*mutex*)) and then decrements *readcount*. If no other readers are waiting to read (*readcount* = 0), it signals that any reader or writer who wishes to access the file may do so (*up*(*wrt*)). Finally, it indicates it is done with *readcount* (*up*(*mutex*)).

line 24 Since the file is not accessed here, we don't need to put semaphores around this part.

lines 25-26 The writer process waits (*down*(*wrt*)) until no other process is accessing the file; it then proceeds to write to the file.

line 27 When the writer is done writing to the file, it signals that anyone who wishes to access the file may do so (*up*(*wrt*)).