READERS AND WRITERS

by

John H. Howard

February 1976 TR-54

The University of Texas at Austin
Department of Computer Sciences
Austin, Texas 78731

INTRODUCTION

This note presents a proof of a monitor (1) for the readers and writers problem (2). The scheduling discipline used is that proposed by Hoare (1), namely FCFS (First-Come-First-Served) except that whenever one reader is started they all are. The proof is based on Howard's method for proving monitors (3) with extensions described below. Both the extensions and the flavor of the results proven may be of some interest.

METHODOLOGY

The proof is given in the appendix. It uses a notation suggested by Owicki (4), in which the assertions are interleaved with the code as comments, in the form*

```
/* precondition */ statement /* postcondition */
```

This provides a felicitous presentation of the relation of the proof to the program, as well as defining the program itself.

The proof schema for monitors in its simplest form is as follows. Let INVAR (main invariant for the monitor) and C(q) (signaling criterion for condition q) be any assertions about the monitor's data. Then if from the assumptions

```
/* INVAR */ q.wait /* C(q) */
/* C(q) */ q.signal /* INVAR */
```

it is possible to prove

```
/* true */ initialization /* INVAR */
/* INVAR */ code for procedure k /* INVAR */
```

then these theorems are unconditionally true of the monitor, and INVAR is satisfied whenever the monitor is inactive.

The following extensions are used here. First, as discussed in (3), it is allowable to "look into" the condition queues with expressions such as q.length and q.first, provided that the proof rule for q.wait is modified to reflect enqueueing and dequeueing

---

* The following typographical conventions are used here.
Comment (assertion) delimiters: /* */
Relational operators: $< > \leq \geq = \neq$
Logical connectives: & | $\Rightarrow$

Second, more specific entry and exit conditions than INVAR may be used. The theorems to be proved may be written

/* INVAR & PRE(k) */ procedure k /* INVAR & POST(k) */

provided that PRE(k) and POST(k) are in a certain sense "local" to the calling process, that is, they cannot be falsified by other calls to the monitor. The notion of what is "local" will be left informal here.

Third, the postcondition of the q.signal operation may be strengthened to be the disjunction of the exit conditions of all the monitor procedures which perform q.wait operations. This may be justified by requiring that signalers be stacked while waiting for their signalees to finish; the signaler regains control of the monitor immediately after his signalee exits. Thus the proof rule for q.signal becomes

/* C(q) */ q.signal /* INVAR & (POST(k1) | ... | POST(kn)) */

where k1...kn are the monitor procedures which contain q.wait operations.

Finally, there is one case in which it is necessary to assume that something "local" to a signaler is preserved across the signal operation (ew=min(ew'+1,sw'+wq'.len) in startread.) There seems to be no great harm in this.

PROPERTIES

The monitor subjected to proof is essentially Hoare's. It has been modified to use history variables directly rather than the differences between them. Thus the number of active readers is (sr-er), the difference between counts of (successful) startread and endread operations. Similarly, (sw-ew) counts active writers. Such use of explicit operation counts aids both in giving calling processes access to the values of such variables and in simplifying the form of the assertions proved. The main properties proved are:

1) Reading and writing are mutually exclusive. This comes from INVAR, which asserts that either sr=er (no readers) or sw=ew (no writers) or both. Furthermore, $sw \leq ew+1$ implies mutual exclusion among writers.

2) No internal deadlocks occur. Again from INVAR, if there is neither an active reader nor an active writer then IDLE holds, in which case all the queues are empty. This was called "no unnecessary waiting" in (3). Note that external deadlocks (involving several monitors) can be dealt with only by considering all called monitors, and the structure of the calling processes, together.

3) When any reader starts, they all do. This comes from

POST(startread), which asserts that rq.len=0. Since a reader starts exactly when he exits from startread, there can be no waiting readers left after any one reader starts.

4) At most one writer precedes any waiting reader. POST(startread) asserts that ew=min(ew'+1,sw'+wq'.len). The primes refer to values at procedure entry (see Manna and Pneuli, (5).) Thus ew≤ew'+1 (at most one writer can finish before a waiting reader starts.) Strict equality holds except for the case that ew'=sw'+wq'.len (this means that there are no writers.)

5) Writers are served first-come-first-served. POST(startwrite) asserts that sw=sw'+wq'.len+1. This means that a waiting writer waits for exactly as many writers as are already in the queue when he requests writing. The +1 term accounts for the waiting writer's own startwrite. Although this does not exactly imply FCFS service, it does imply waiting times equivalent to those of FCFS. Since readers operate in parallel, no attempt is made to derive similar results bounding the number of reads performed while a writer waits.

6) Dropping internal information from the procedures' pre- and post-conditions, but retaining reference to the history variables sr, er, sw, and ew, the functional properties derived for the monitor are:

```
/* true */ startread /* sr>er & sw=ew */
/* sr>er */ endread /* true */
/* true */ startwrite /* sw=ew+1 & sr=er */
/* sw=ew+1 */ endwrite /* true */
```

It is up to the callers to satisfy the preconditions, using their knowlege of the postconditions.

REFERENCES

1. Hoare, C.A.R., Monitors: an operating system structuring concept. Comm ACM 17,10 (October 1974), 549-557.

2. Courtois, P.J., Heymans, F., and Parnas, D.L., Concurrent control with readers and writers. Comm ACM 14,10 (October 1971), 667-668.

3. Howard, J.H., Proving monitors. Comm ACM 19,4 (May 1976), forthcoming.

4. Owicki, S. and Gries, D., Verifying properties of interactive programs: an axiomatic approach. Comm ACM 19,4 (May 1976), forthcoming.

5. Manna, Z. and Pneuli, A., Axiomatic approaches to total correctness of programs. Acta Informatica 3 (1974), 243-263.

APPENDIX

```
readers and writers: monitor
begin

sr,sw: integer;          /* starts */
er,ew: integer;          /* ends */
rq,wq: condition;        /* queues */

/*****************************************************

Main invariant:
  INVAR:  (IDLE | READING | WRITING) & RP(0) & WP(1)

Preconditions for signal operations:
  C(rq):  sr≥er & sw=ew & rq.len>0 & RP(1) & WP(1)
  C(wq):  sr=er & sw=ew & wq.len>0 & RP(0) & WP(1)

where common sub-assertions are defined by:
  IDLE:  sr=er & sw=ew & rq.len=0 & wq.len=0
  READING:  sr>er & sw=ew & (rq.len=0 | wq.len>0)
  WRITING:  sr=er & sw=ew+1
  RP(K=0..1):  for all J=1..rq.len do with data(rq(J)) do
                    ew=ew'+K & ew'<sw'+wq'.len od od
  WP(K=0..1):  for all J=1..wq.len do with data(wq(J)) do
                    sw=sw'+wq'.len-J+K od od
Primes refer to values at procedure entry.  WP(0) will be used
later.  RP and WP are satisfied vacuously for empty queues.

Preconditions of wait operations are INVAR modified to reflect the
caller entering at the tail of the queue:
  PRE(rq.wait):          (sr>er&sw=ew&wq.len>0 | sr=er&sw=ew+1) &
                         RP(0) & WP(1) & ew=ew'+0 & ew'<sw'+wq'.len
  PRE(wq.wait):          (sr>er&sw=ew | sr=er&sw=ew+1) & RP(0) &
                         WP(1) & sw=sw'+wq'.len-(wq.len+1)+1

Postconditions of wait operations are the corresponding signal
preconditions, modified to reflect the caller leaving the head of
the queue and the remainder of the queue shifting up:
  POST(rq.wait):         sr≥er & sw=ew & RP(1) & WP(1) & ew=ew'+1 &
                         ew'<sw'+wq'.len
  POST(wq.wait):         sr=er & sw=ew & RP(0) & WP(0) &
                         sw=sw'+wq'.len-1+1

Postconditions of signal operations are the exit conditions of the
procedures containing corresponding wait operations:
  POST(rq.signal)=POST(startread):
                         READING & rq.len=0 & WP(1)
  POST(wq.signal)=POST(startwrite):
                         WRITING & RP(0) & WP(1)

*****************************************************/
```

```
procedure startread;
/* INVAR */
begin /* (IDLE|READING|WRITING) & RP(0) & WP(1) & ew=ew' & sw=sw'
                          & wq.len=wq'.len */
  if sw+wq.len>ew then
    /* (sr>er&sw=ew&wq.len>0 | sr=er&sw=ew+1) & RP(0) & WP(1) &
                          ew=ew' & ew'<sw'+wq'.len */
    rq.wait
    /* sr≥er & sw=ew & RP(1) & WP(1) & ew=ew'+1 & ew'<sw'+wq'.len
                          */
  else
    /* sr≥er & sw=ew & rq.len=0 & wq.len=0 & ew=ew' &
                          ew'=sw'+wq'.len */
  fi;
  /* sr≥er & sw=ew & RP(1) & WP(1) & ew=min(ew'+1,sw'+wq'.len) */
  sr:=sr+1;
  /* sr>er & sw=ew & RP(1) & WP(1) & ew=min(ew'+1,sw'+wq'.len) */
  if rq.len>0 then
    /* sr>er & sw=ew & rq.len>0 & RP(1) & WP(1) &
                          ew=min(ew'+1,sw'+wq'.len) */
    rq.signal
    /* READING & rq.len=0 & WP(1) & ew=min(ew'+1,sw'+wq'.len) */
  else
    /* READING & rq.len=0 & WP(1) & ew=min(ew'+1,sw'+wq'.len) */
  fi;
  /* READING & rq.len=0 & WP(1) & ew=min(ew'+1,sw'+wq'.len) */
end startread;
/* INVAR & sr>er & rq.len=0 & ew=min(ew'+1,sw'+wq'.len) */

procedure endread;
/* INVAR & sr>er */
begin /* READING & RP(0) & WP(1) */
  er:=er+1;
  /* sr≥er & sw=ew & (rq.len=0|wq.len>0) & RP(0) & WP(1) */
  if (sr=er)&(wq.len>0) then
    /* sr=er & sw=ew & wq.len>0 & RP(0) & WP(1) */
    wq.signal
    /* WRITING & RP(0) & WP(1) */
  else
    /* (IDLE | READING) & RP(0) & WP(1) */
  fi;
  /* (IDLE | READING | WRITING) & RP(0) & WP(1) */
end endread;
/* INVAR */
```

```
procedure startwrite;
/* INVAR */
 begin /* (IDLE | READING | WRITING) & RP(0) & WP(1) & sw=sw' &
                                wq.len=wq'.len */
   if sr+sw>er+ew then
     /* (READING | WRITING) & RP(0) & WP(1) & sw=sw' &
                                wq.len=wq'.len */
     wq.wait
     /* sr=er & sw=ew & RP(0) & WP(0) & sw=sw'+wq'.len */
   else
     /* IDLE & sw=sw'+wq'.len */
   fi;
   /* sr=er & sw=ew & RP(0) & WP(0) & sw=sw'+wq'.len */
   sw:=sw+1;
   /* WRITING & RP(0) & WP(1) & sw=sw'+wq'.len+1 */
 end startwrite;
 /* INVAR & sw=ew+1 & sw=sw'+wq'.len+1 */

 procedure endwrite;
 /* INVAR & sw>ew */
 begin /* WRITING & RP(0) & WP(1) */
    ew:=ew+1;
    /* sr=er & sw=ew & RP(1) & WP(1) */
    if rq.len>0 then
      /* sr=er & sw=ew & rq.len>0 & RP(1) & WP(1) */
      rq.signal
      /* READING & rq.len=0 & WP(1) */
    else if wq.len>0 then
      /* sr=er & sw=ew & rq.len=0 & wq.len>0 & WP(1) */
      wq.signal
      /* WRITING & RP(0) & WP(1) */
    else
      /* IDLE */
    fi;
    /* (IDLE | READING | WRITING) & RP(0) & WP(1) */
 end endwrite;
 /* INVAR */

 sr:=sw:=er:=ew:=0;
 /* INVAR */

 end readers and writers;
```