Monitor DB_RW_Bounded

Const
Rmax : integer; // max consecutive reader sessions

Var
readCount : integer; // readers inside
writerActive : boolean; // a writer inside?
waitingWriters : integer; // writers waiting
readerSessions : integer; // consecutive reader sessions
turnReaders : boolean; // true ⇒ readers' turn
canRead, canWrite : condition;

Procedure StartRead;

Begin
While (writerActive)
OR ((NOT turnReaders) AND (waitingWriters > 0)) do
canRead.wait;

readCount := readCount + 1;
End;

Procedure EndRead;

Begin
readCount := readCount - 1;

If readCount = 0 then // end of a reader session
Begin
readerSessions := readerSessions + 1;

```
If (readerSessions >= Rmax) AND (waitingWriters
> 0) then
Begin
  turnReaders := false;    // give turn to
writers
  readerSessions := 0;
  canWrite.signal;
End
else
  If waitingWriters > 0 then
    canWrite.signal
  else
    canRead.signal;
```

End;
End;

Procedure StartWrite;

Begin
waitingWriters := waitingWriters + 1;

While (writerActive) OR (readCount > 0) OR
(turnReaders) do
canWrite.wait;

waitingWriters := waitingWriters - 1;
writerActive := true;
End;

begin

readCount := 0;
writerActive := false;
waitingWriters := 0;
readerSessions := 0;
turnReaders := true; // readers allowed first

end

====end monitor

Procedure EndWrite;

Begin
writerActive := false;
turnReaders := true; // return turn to readers
readerSessions := 0;

canRead.signal; // wake readers if any
If waitingWriters > 0 then
canWrite.signal;
End;

Processes

```
Process P_Reader_i;
Begin
Call DB_RW_Bounded.StartRead;
ReadDatabase();
Call DB_RW_Bounded.EndRead;
End;

Process P_Writer_j;
Begin
Call DB_RW_Bounded.StartWrite;
WriteDatabase();
Call DB_RW_Bounded.EndWrite;
End.
```