Published: 06/08/66

Identification

Standard Interlock Mechanism
C.A.Cushing

Purpose

The interlock mechanism described here is used for all data bases in the basic file system which are common to more than one process. This mechanism is used to some degree in manipulating the following data bases:

1. System Segment Tables (PST,DST,AST)
2. Active File Table (AFT)
3. I/O Queues (Q)
4. Core Map
5. All directories
6. Process waiting tables (PWT)

Introduction

Each data base for which the standard interlock mechanism is used must have the following three consecutive words in the data base.

1. LOCK

This word initially has the value zero and is used to lock the data base from all other processes. If the contents of lock are 0, the data base is unlocked. If the contents are non-zero, the data base is locked on behalf of the process whose identification number is the non-zero value of lock.

2. NO-MORE-READERS SWITCH (NOMORE)

If this switch is on, no processes will be allowed to read this data base. This switch is set on by a process which is about to go blocked waiting to modify the data base. Whenever lock or read count change from non-zero to zero, this switch is checked by the process which caused the change. If the switch is on, it is turned off and this process calls notify to unblock any processes waiting to use the data base.

3. READ COUNT

This word contains a count of the number of processes currently using the data base for reading purposes only.

Interlocking Primitives

1. If a process wishes to modify a common data base, the subroutine

```
call modlock (p,event,var,waitrtn);
```

tests the lock of the data base pointed to by the pointer p (ITS pair) to see if modification can be permitted. If this process may not modify the data base, the no-more-readers switch is set on, the process is entered on the PWT according to event and var, and control from modlock is returned to the statement labeled waitrtn in the calling procedure. At this point, the calling procedure has the opportunity of doing any necessary processing before going blocked.

IMPLEMENTATION

```
            segdef    modlock
            segref    processdata,processid
            segref    pwn,addpwt
            segref    pwn,delpwt
            temp      tryct
            tempd     arglist (3)       area for argument list
modlock:    save
            eapbp     ap|2,*
            eapbp     bp|0,*            bp=ptr to lock of data base
            fld       =4b25,dl          store 2* number
            staq      arglist           of arguments as first pair
            ldaq      ap|4
            staq      arglist+2         (event)
            ldaq      ap|6
            staq      arglist+4         (var)
            stz       tryct
            lda       processid
retry:      stac      bp|0              try to lock data base
            tnz       wait2
            szn       bp|2              are readers in data base?
            tnz       wait1
            szn       tryct             no.test number of tries
            tze       rtn               successful on first try
            call      delpwt(arglist)   successful on second try
rtn:        return

wait1:      stz       bp|0              readers in data base-unlock
wait2:      szn       tryct             data base already locked
            tnz       waitrtn
            call      addpwt (arglist)  try again but first get on PWT
```

```
          aos       tryct          in case of second
          tra       retry          failure

waitrtn:  aos       bp|1           failure-do not let any
          ldaq      ap|8,*         others succeed
          ldb       sp|16,*
          ldi       sp|21
          staq      sp|20
          lreg      sp|8
          tra       sp|20,*        return to wait
          end
```

2) If a process wishes to read a common data base, the subroutine

```
call readlock (p,event,var,waitrtn);
```

must be called. A normal return implies that the process has successfully been added to the number of readers of the data base and may read it also. A return to waitrtn implies either that the data base is locked by another process or that another process is blocked waiting for readers to leave this data base (i.e., nomore-readers switch is on). This process must now go blocked or remove itself from event list in the PWT.

IMPLEMENTATION

```
          segdef    readlock
          segref    processdata,processid
          segref    pwn,addpwt
          segref    pwn,delpwt
          temp t    tryct
          tempd     arglist (3)

readlock: save
          eapbp     ap|2,*
          eapbp     bp|0,*              bp=ptr to data base lock
          fld       =4b25.dl            2* number of arguments
          staq      arglist
          ldaq      ap|4
          staq      arglist+2           (event)
          ldaq      ap|6
          staq      arglist+4           (var)
          stz       tryct
          lda       processid
retry:    stac      bp|0                try to lock data base
```

```
            tnz       wait2              if locked,
            szn       bp|1               are more readers
                                         allowed?
            tnz       wait1
            szn       tryct              yes. test number of tries
            tze       rtn                successful on first try
            call      delpwt(arglist)    successful on second try

rtn:        aos       bp|2               increase read count
            stz       bp|0               unlock data base
            return


wait1:      stz       bp|0               unlock data base, no
                                         more readers allowed
wait2:      szn       tryct              data base locked
            tnz       waitrtn
            call      addpwt(arglist)    try again, but first
            aos       tryct              get on PWT
            tra       retry              in case of second failure

waitrtn:    ldaq      ap|8,*             failure
            ldb       sp|16,*
            ldi       sp|21
            staq      sp|20
            lreg      sp|8
            tra       sp|20,*            return to wait
            end
```

3. When a process has finished modifying a common data base, the subroutine

```
call unlock (p,event,var,errtn);
```

unlocks the data base for this process and wakes up any other processes which may have been blocked trying to use the data base. If the process calling unlock was not the process which locked the data base, an error will be reflected.

IMPLEMENTATION

```
            segdef    unlock
            segref    processdata, processid
            segref    pwn,notify
            tempd     arglist(3)

unlock:     save
            eapbp     ap|2,*
            eapbp     bp|0,*             bp=ptr to lock of data
                                         base
```

```
          fld       =4b25,dl
          staq      arglist
          ldaq      ap|6
          staq      arglist+2          (event)
          ldaq      ap|6
          staq      arglist+4          (var)
          lda       processid          be sure process unlocking
          cmpa      bp|0               base originally locked it

          tnz       error
          stz       bp|0               unlock data base
          call      notify(arglist)    wake up processes blocked

rtn:      return                       because of lock

error:    ldaq      ap|8,*             logic error
          ldb       sp|16,*
          ldi       sp|21
          staq      sp|20
          lreg      sp|8
          tra       sp|20,*            error return
          end
```

4. When a process is through reading a common data base, the subroutine

```
call decrease (p,event,var);
```

must be called.

IMPLEMENTATION

```
          segdef    decrease
          segref    pwn,notify
          tempd     arglist(3)


decrease: save
          eapbp     ap|2,*
          eapbp     bp|0,*
          lda       =-1                reduce number of readers
          asa       bp|2               by one
          tnz       rtn                if last reader

last:     stz       bp|1               zero no-more-readers switch
          fld       =4b25,dl
          staq      arglist
          ldaq      ap|4
          staq      arglist+2
```

```
          ldaq      ap|6
          staq      arglist+4
          call      notify(arglist)   and wake up processes
                                      blocked because of readers
rtn:      return
          end
```

Usage

For example, a process may attempt to modify a data base with the following PL/I sequence

```
try:      call modlock (p,event,var,waitrtn);
modify:        .
               .
               .
waitrtn:  call block;        /* wait for data base to become
                                unlocked*/
          go to try;
```

or attempt to read a data base with the following PL/I sequence

```
try:      call readlock (p,event,var,waitrtn);
read:          .
               .
               .
waitrtn:  call block;
          go to try;
```

or unlock a data base with the following sequence

```
removelock:  call unlock (p,event,var,err);
continue:         .
                  .
                  .
err:    call logic_error_handler;
```

Data Base Entry Interlocks

In some cases, it may suffice to lock a particular entry in a common data base rather than the entire data base. This is

done when it is necessary to make a change to an entry in a data base which does not affect the rest of the data base, e.g., the change does not affect the length of the entry. It may not only suffice but also be advisable to lock an entry in a data base rather than the entire data base. This is true when the change to be made to an entry may take an undetermined amount of time, and this change is dependent on the current contents of the entry. For example, this lock is applied to a branch in a directory from the time the branch is found by segment control until the file to which it points is activated by usage control.

For every data base in which this technique is used, each entry in that data base must contain the following additional word of information.

LOCK - This word is used in the same manner as the lock for the entire data base as previously described.

Entry Interlocking Primitives

If a process wishes to lock an entry in a common data base (in order to modify it or read a stable copy of it), the process must first read the data base to find the entry if the location of the entry is not already known. Once the location of the entry is known, the process must record its identification in the lock for that entry. This is done by the subroutine

```
call entrylock (ep,event,var,waitrtn);
```

where ep is a pointer (ITS pair) to the lock of the entry in the data base. If entrylock is or is not successful in locking the entry in the data base for this process, the process must leave the data base as a reader, i.e., decrease the read count of the data base.

IMPLEMENTATION

```
            segdef   entrylock
            segref   processdata, processid
            segref   pwn,delpwt
            segref   pwn,addpwt
            temp     tryct
            tempd    arglist(3)

entrylock:  save
            eapbp    ap|2,*
            eapbp    bp|0,*              bp=ptr to lock in entry
```

```
          fld       =4b25,dl            2* number of arguments
          staq      arglist
          ldaq      ap|4
          staq      arglist+2           event
          ldaq      ap|6
          staq      arglist+4           var
          stz       tryct
          lda       processid

retry:    stac      bp|0                try to lock entry
          tnz       wait
          szn       tryct
          tze       rtn
          call      delpwt(arglist)     successful on second try

rtn:      return

wait:     szn       tryct               unsuccessful
          tnz       waitrtn
          call      addpwt(arglist)     try again but first
          aos       tryct               get on PWT in case
          tra       retry               of second failure

waitrtn:  ldaq      ap|8,*              failure
          ldb       sp|16,*
          ldi       sp|21
          staq      sp|20
          lreg      sp|8
          tra       sp|20,*             return to wait
          end
```

When the entry is to be unlocked, the subroutine

```
call entryunlock (ep,event,var,err);
```

unlocks the entry for the process and wakes up any processes which are waiting for it to become unlocked.

IMPLEMENTATION

```
          segdef    entryunlock
          segref    processdata,processid
          segref    pwn,notify
          tempd     arglist(3)

entrylock:save
          eapbp     ap|2,*
          eapbp     bp|0,*              bp=ptr to entry lock
```

```
          fld       =4b25,dl
          staq      arglist
          ldaq      ap|4
          staq      arglist+2
          ldaq      ap|6
          staq      arglist+4
          lda       bp|0                 be sure entry locked
          cmpa      processid            by this process
          tnz       err
          stz       bp|0                 unlock entry
          call      notify(arglist)      wake up processes
                                         blocked
          return                         because of lock

err:      ldaq      ap|8,*               logic error
          ldb       sp|16,*
          ldi       sp|21
          staq      sp|20
          lreg      sp|8
          tra       sp|20.*              error return
          end
```

Usage

For example, a process may attempt to modify an entry in a common data base with the following PL/I sequence

```
          /* If location of entry not known, begin here */

tryread:  call readlock(p,event,var,waitrtn2);
search:   search_switch="1"b;

          /*search for entry in data base*/
               .
               .
               .
               .
found:    go to trylock;

          /*If location of entry known,begin here*/

trymodify: search_switch="0"b;
trylock:  call entrylock (ep,event1,var1,waitrtn1);
          if search_switch then
               call decrease (p,event,var);

          /*entry locked,can modify fixed-length items in entry*/
```

```
modify_fixed:
                    .
                    .
                    .
          /*must lock data base to modify variable-length
            items in entry*/

modify_var: call modlock (p,event,var,waitrtn3);
                    .
                    .
                    .
                    .
done:     call unlock (p,event,var,err);
          call entryunlock (ep,event1,var1,err);
                    .
                    .
                    .
                    .
waitrtn1:  if search_switch then call decrease(p,event,var);
waitrtn2:  call block;
           if search_switch then go to tryread;
           else go to trylock;
waitrtn3:  call block;
           go to modify_var;
```

Special Interlock

The procedures addpwt and delpwt must handle interlocking of the wired-down PWT in a special way. Those procedures which must be entered into (deleted from) an event list in the wired-down PWT, must continuously attempt to lock the list in order to be entered (deleted), i.e., they must loop.

```
     call looplock (p);
```

IMPLEMENTATION

```
          segdef    looplock
          segref    processdata, processid

looplock: save
          eapbp     ap|2,*       bp=ptr to lock of event
          eapbp     bp|0,*       list in latched PWT
          lda       processid
          stac      bp|0         attempt to lock list
          tnz       *-1
          return                 locked
          end
```

When the attempt is finally successful, the process can then be entered or deleted. The following subroutine unlocks the list when the task is completed.

```
call loopunlock (p,err);
```

IMPLEMENTATION

```
                segdef      loopunlock
                segref      processdata,processid
loopunlock:     save
                eapbp       ap|2,*
                eapbp       bp|0,*
                lda         bp|0        be sure process unlocking list
                cmpa        processid   was process that locked it
                tnz         err
                stz         bp|0        unlock list
                return

err             ldaq        bp|4,*      logic error
                ldb         sp|16,*
                ldi         sp|21
                staq        sp|20
                lreg        sp|8
                tra         sp|20,*
                end
```