A SIMULATOR FOR DETERMINING THE PERFORMANCE OF
TRANSACTION MANAGER AND LO...(U) ELECTRONIC SYSTEMS DIV
HANSCOM AFB MA D A VARVEL ET AL. 31 AUG R4
UNCLASSIFIED FSD TR 84 195
A Simulator for Determining the Performance of Transaction Manager and Lock Manager Combinations in a Database

DONALD A. VARVEL
WILLIAM PERRIZO

31 August 1984

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

Prepared for
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
DEPUTY FOR DEVELOPMENT PLANS
HANSCOM AIR FORCE BASE, MASSACHUSETTS 01731
LEGAL NOTICE

When U.S. Government drawings, specifications or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

OTHER NOTICES

Do not return this copy. Retain or destroy.

REVIEW AND APPROVAL

This technical report has been reviewed and is approved for publication.

THOMAS SELINKA, Capt., USAF
Project Officer

SILVIO V. D'ARCO, Lt Col., USAF
Deputy Director, Tactical C'T Systems Planning
Deputy for Development Plans

FOR THE COMMANDER

DONALD L. MILLER, Colonel, USAF
Assistant Deputy for Development Plans
A Simulator for Determining the (Cont.)

Most database systems use locking for concurrency control. Responsiveness is degraded when transactions spend much time waiting for locks. In those situations in which the lockable units need not be processed in a particular order, differences in the order of processing can make large differences in the durations of the transactions, i.e., responsiveness. Order of processing may be modified by the use of a combination of non-blocking and potentially blocking lock requests. A simulation is used to investigate the performance of several such algorithms in a variety of settings. 

Keywords include:
Block 11 Continued

Performance of Transaction Manager and Lock Manager Combinations in a Database.
(Unclassified)
### TABLE OF CONTENTS

<table>
<thead>
<tr>
<th></th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2. Algorithms</td>
<td>4</td>
</tr>
<tr>
<td>3. The Simulation</td>
<td>9</td>
</tr>
<tr>
<td>4. Summary and Conclusions</td>
<td>11</td>
</tr>
<tr>
<td>5. Bibliography</td>
<td>13</td>
</tr>
<tr>
<td>6. Appendix A: The Program</td>
<td>15</td>
</tr>
<tr>
<td>7. Appendix B: Results</td>
<td>39</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Accession For</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTIS GRAAI</td>
</tr>
<tr>
<td>DTIC TAB</td>
</tr>
<tr>
<td>Unannounced</td>
</tr>
<tr>
<td>Justification</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>By</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distribution/</td>
</tr>
<tr>
<td>Availability Codes</td>
</tr>
<tr>
<td>Avail and/or</td>
</tr>
<tr>
<td>Dist</td>
</tr>
<tr>
<td>Special</td>
</tr>
<tr>
<td>A-1</td>
</tr>
</tbody>
</table>

iii
<table>
<thead>
<tr>
<th>TABLE</th>
<th>DESCRIPTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Accessing the Lockable Units in the Natural Order, U₁, U₂, U₃, U₄, U₅</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>The Alternative Order U₁, U₃, U₄, U₅, U₂</td>
<td>3</td>
</tr>
</tbody>
</table>
1. Introduction

The information system supporting distributed command and control (C²) operations may be viewed as a distributed database. This is an attractive view because the distributed database literature includes solutions to some of the problems facing the implementor of a distributed C² operation. General discussions of distributed database may be found in [CERI84] and [DATE83], while methods applicable to C² site initialization, recovery, and back-up are discussed in [ATTA83].

Database techniques have been developed largely for a commercial environment where responsiveness is less important than complete accuracy, and short breaks in service may be tolerated. For example, concurrency control may involve locking units of information for extended periods of time, so that the information is not generally available. If transactions are kept short, as recommended in [DATE83], the disruption is not great. [ATTA83], however, outlines solutions to the problems of initialization and recovery that necessarily involve huge transactions.

We have therefore addressed the problem of responsiveness in a locking environment.

Many database transactions, and parts of nearly all, can be viewed as a set operations; this is particularly explicit in the relational model [CODD70]. Sets need not be accessed in any particular order, but on a sequential machine some order must be chosen. Usually that order is arbitrary. We propose to modify that order to increase responsiveness.
The order of access to data items becomes important in the event of conflict. Conflict occurs when one transaction attempts to access a data item on which another transaction holds a lock that is not compatible with the attempted access. In that case, the requesting transaction enters a nonbusy wait state. Barring deadlock it will return to the ready state when the requested lock becomes available.

Deadlock occurs when $T_1$ waits for a resource held by $T_2$ and $T_2$ waits, possibly indirectly, for a resource held by $T_1$. The situation in which the requesting transaction either receives a lock or waits assumes some means of deadlock detection and recovery. Our simulation assumes such a system, but we do not model the actual effects of deadlock.

Transaction $T_1$ must access five lockable units of data, $U_1-U_5$. Each lockable unit requires 10 time units to process. Assume that locks for $U_1$ and $U_3-U_5$ are available, and that the lock request for each requires one time unit. $U_2$, however, is locked and will remain locked until 50 time units after $T_1$ starts. Accessing the lockable units in the natural order, $U_1$, $U_2$, $U_3$, $U_4$, $U_5$, proceeds as in Table 1.
<table>
<thead>
<tr>
<th>Activity</th>
<th>Time Required</th>
<th>Time When Complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock U₁</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Process U₁</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>Lock U₂</td>
<td>39</td>
<td>50</td>
</tr>
<tr>
<td>Process U₂</td>
<td>10</td>
<td>60</td>
</tr>
<tr>
<td>Lock U₃</td>
<td>1</td>
<td>61</td>
</tr>
<tr>
<td>Process U₃</td>
<td>10</td>
<td>71</td>
</tr>
<tr>
<td>Lock U₄</td>
<td>1</td>
<td>72</td>
</tr>
<tr>
<td>Process U₄</td>
<td>10</td>
<td>82</td>
</tr>
<tr>
<td>Lock U₅</td>
<td>1</td>
<td>83</td>
</tr>
<tr>
<td>Process U₅</td>
<td>10</td>
<td>93</td>
</tr>
</tbody>
</table>

Table 1

The alternative order U₁, U₃, U₄, U₅, U₂ proceeds as in Table 2.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Time Required</th>
<th>Time When Complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock U₁</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Process U₁</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>Lock U₂</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>Process U₂</td>
<td>10</td>
<td>22</td>
</tr>
<tr>
<td>Lock U₃</td>
<td>1</td>
<td>23</td>
</tr>
<tr>
<td>Process U₃</td>
<td>10</td>
<td>33</td>
</tr>
<tr>
<td>Lock U₄</td>
<td>1</td>
<td>34</td>
</tr>
<tr>
<td>Process U₄</td>
<td>10</td>
<td>44</td>
</tr>
<tr>
<td>Lock U₅</td>
<td>6</td>
<td>50</td>
</tr>
<tr>
<td>Process U₅</td>
<td>10</td>
<td>60</td>
</tr>
</tbody>
</table>

Table 2

This example shows that even in a very simple case the order of processing can have a substantial effect on the amount of real time required to process a given transaction. We have devised several means of improving the order of access to data items to avoid long waits caused by locking.

In Section 2 we outline three new look managers and several algorithms for using them. In section 3 we describe and justify a simulation of combinations of look managers and protocols from Section 2. Section 4 consists of a summary and conclusions. The simulation program is
included as Appendix A, and the results as Appendix B.

2. Algorithms

In this section we present some new look managers and the transaction managers that use them. First we define the context in which we will operate.

We will assume, for the sake of this discussion, a centralized (non-distributed) database that uses locks and deadlock detection.

A concurrent database system must have lock manager, which we abbreviate LM. LMO below is an example. Transactions get access to data by requesting locks from the lock manager, and return the resources to the system by informing the LM of the release. The lock manager may cause transactions to become blocked or unblocked. It maintains queues for those data items that are requested while locked.

The acquisition portion of the usual lock manager may be viewed as the following:

```
Procedure LMO(Lockable_Unit_ID, Lock_Level, Transaction_ID);
Begin
  If Lockable_Unit_ID is available at Lock_Level then
    Record look for Transaction_ID
  Else begin
    Place Transaction_ID in queue for Lockable_Unit_ID;
    Block Transaction_ID;
    Cause system to dispatch another transaction
  End
End;
```

A transaction manager (TM) is an entity that interprets high-level queries by issuing lower-level requests, including negotiations with the
lock manager. Each transaction has a logical transaction manager, although the actual code might be shared. TM0 is an example.

Here is how the usual transaction manager deals with a series of data items $D_1, \ldots, D_n$, which may be processed in any order:

Transaction manager TM0:

```plaintext
Begin
  For i := 1 to n do begin
    LMO(D_i, Lock_Level, Trans_ID);
    Process D_i
  End
End;
```

A given transaction running on a given database at a particular time will find certain data items (which we will refer to as lockable units or LU's) available at certain times but not at others. In our simulation we present this as a fixed background. Queues of fairly stable length tend to form waiting for high-activity data items. Our simulation includes both lockable units with conflicts at certain times and LU's with fixed-length queues. We also simulate processing delays for the various lockable units.

The time required to acquire an available lock is taken as the unit. This lock-request delay is chosen as (relatively) large as it is on the assumption that the lock manager will at least occasionally have an entry queue.

LMO is not useful in determining whether or not to process an LU immediately. A more useful lock manager, LM1, would have two lock-request entry points: one blocking and one non-blocking. The blocking request is as LMO. The non-blocking request returns True if
False, depending on whether the lock is presently available. This is
LM1, assuming LMO is still available as a blocking entry point:

Function LM1(Lockable_Unit_ID, Lock_Level, Transaction_ID): Boolean;
Begin
  If Lockable_Unit_ID is available at Lock_Level then begin
    Record lock for Transaction_ID;
    LM1 := True
  End
  Else LM1 := False
End;

In LM2, if the lock is not available, the transaction is entered into
its queue. This requires slight changes in the blocking and release
parts of the lock manager: the transaction at the head of the queue may
not be blocked, and the requesting transaction may already have the
lock.

Function LM2(Lockable_Unit_ID, Lock_Level, Transaction_ID): Boolean;
Begin
  If Lockable_Unit_ID is already locked by Transaction_ID then
    LM2 := True
  Else if Lockable_Unit_ID is available at Lock_Level then begin
    Record lock for Transaction_ID;
    LM2 := True
  End
  Else begin
    If Transaction_ID is not in Lockable_Unit_ID's queue then
      Place Transaction_ID in Lockable_Unit_ID's queue;
    LM2 := False
  End
End;

It is possible to propose a variety of algorithms that use LM1 or LM2.
We define and simulate three such algorithms.

Our first TM makes blocking requests only when an entire pass through
the LU's using non-blocking requests produces no results. It may be
used with either LM1 or LM2 (as may TM2 and TM3).

Transaction manager TM1:
Begin
While unprocessed units remain do begin
   Repeat process list using nonblocking requests
   Until a pass finds all unprocessed units locked;
   If at least one unprocessed unit remains then
      Make a blocking request for an unprocessed unit
   End
End;

The second TM makes a pass through the list of LU's using nonblocking requests and then arbitrarily selects an unprocessed LU to wait for; these two actions are alternated until no LU's remain unprocessed. It differs from TM1 in that it does not wait for an entire unsuccessful pass before issuing a blocking request.

Transaction manager TM2:
Begin
   While unprocessed LU's remain do begin
      Process list of LU's using nonblocking requests;
      If at least one unprocessed LU remains then
         Pick an unprocessed LU and make a blocking request for it
      End
   End;
End;

TM3 makes still fewer nonblocking requests. It makes a nonblocking pass and then a blocking pass. It takes what is immediately available, than waits for what is not.

Transaction manager TM3:
Begin
   Process list of LU's using nonblocking requests;
   Process list of LU's using blocking requests
End;

Transaction managers TM1, TM2, and TM3 all eventually choose arbitrary lockable units for which to issue blocking lock requests; in the simulation it is the first unprocessed LU. The one arbitrarily chosen may not be the best. Another LU may become available much earlier. We would like always to be sure of selecting the best one. Using the lock managers discussed so far, that cannot be done.
To do so we must make more substantial modifications to the lock manager. We need a lock manager that can be given a list of requests and instructed to unblock the transaction and return the lockable unit's ID when any of the lockable units become available.

LM3 has three parts: LM3I (Initial) for nonblocking requests, LM3W (Wait) for multiple blocking requests, and LM3R (Release) for releasing locks. (Each of the previous lock managers also must have a release portion, but all have been so similar to LMO's that we have omitted discussing them.) LM3 generates the blocking request list in LM3I.

Function LM3I(Lockable_Unit_Id,Lock_Level,Transaction_ID):Boolean;
Begin
    If Lockable_Unit_ID is already held by Transaction_ID then
        LM3I := True
    Else if Lockable_Unit_ID is available at Lock_Level then begin
        Record lock for Transaction_ID;
        LM3I := True
    Else begin
        If Transaction_ID is not in Lockable_Unit_ID's queue then
            Place Transaction_ID in Lockable_Unit_ID's queue;
        If Or list for Transaction_ID does not exist then
            Create an Or list for Transaction_ID;
        Insert Lockable_Unit_ID in Or list;
        LM3I := False
    End
End;

Function LM3W(Transaction_ID) : Lockable_Unit_ID_type;
Begin
    If Or list is empty or non-existent then LM3W := error
    Else if no lock in the Or_list is available then begin
        Block Transaction_ID;
        Dispatch another transaction
    End
    Else begin
        Select an awarded lock;
        Remove that lock from the Or_list;
        LM3W := Lockable_Unit_ID
    End
End;
Procedure LM3R(Lockable_Unit_ID);
Begin
If Lockable_Unit_ID's queue is nonempty then begin
  Award the lock to a transaction from the queue;
  If transaction is waiting then begin
    Remove that lock from the Or_list;
    Place the Lockable_Unit_ID where the transaction expects a
    return value from LM3W;
    Unblock the transaction
  End
End
End;

Transaction manager TM4:
Begin
  Make a pass through the list using LM3I;
  While at least one unprocessed unit remains do
    Process(LM3W(Transaction_ID))
End;

3 The simulation
In this section we describe our simulation.

An estimate of performance will help determine the relative worth of the
various methods. We chose to perform a Monte Carlo simulation in order
to produce reasonable answers in reasonable time.

Transactions are generated at random within supplied parameters. A
simulation of each Transaction Manager/Lock Manager combination is run
against each generated transaction. Because the algorithms run against
the same transactions, the results are comparable. The cycle of
transaction generation and running of algorithms is repeated -- here,
twenty times -- and the results averaged.

We have not modeled deadlock. Some authors [KUNG81, GRAY81] have
maintained that deadlock is rare. In any case, we maintain that deadlock would not substantially change the relative results.

We wished to have a standard against which to compare each of the methods, including the usual TM0. We have defined an optimal time and have computed it for each transaction. We have compared each of the algorithms to that standard.

For information on the computation of optimal time, see [VARV84].

The two statistics we record are time active and lock requests. Minimum lock requests is achieved by TM0. The other transaction/lock manager combinations reduce time active at the cost of some additional lock requests.

We assume that a few extra lock requests may be tolerated rather well, but that a doubling of lock requests should purchase very substantial reductions of time active. Accordingly, our evaluation function is

\[ (1 + (1 - R/U)^2) T \]

where \( T \) is time active, \( R \) is number of lock requests, and \( U \) is lockable units.

To put these numbers in perspective, we have displayed the ratio of each method's evaluation to optimal.

Each simulation determines the mean behavior of each of the algorithms running twenty transactions. Twelve simulations were run, representing
the cross product of 5, 10, 15, and 20 lockable units with light, medium, and heavy activity.

The two probabilities in the setup parameters are not comparable. The second is each lockable unit's probability of having a queue. The first, however, is the probability of an adverse request in a given ten-time-unit interval. We used those two numbers to specify level of activity.

The results of the simulation are given in Appendix B.

4 Summary and conclusions
A transaction may wait for one locked unit while it could be processing others that are not locked. Worse, some of the other units could become locked in the interim. We have presented several approaches to solving that problem.

We have devised three modified lock managers, which we call LM1, LM2, and LM3. LM1 and LM2 may be used with any of the transaction managers TM1, TM2, and TM3, but LM3 is used only with TM4; thus, we present seven combinations.

The performance of these combinations has been simulated and compared with optimal performance. Each combination was tested with transactions of various sizes and with several levels of conflict.

4.1 Conclusions
No combination of transaction and lock managers has enough information to achieve optimal performance. We were able to simulate optimality only because all conflicts were known in advance. TM4/LM3 achieved the best performance in our simulation, but might prove difficult to implement. TM3/LM2 performed almost as well as TM4/LM3 and might constitute a good practical choice.

We believe that either TM3/LM2 or TM4/LM3 should be considered for implementation in database systems where response time is critical.
BIBLIOGRAPHY


[CODD70]: Codd, E.F. "A Relational Model for Large Shared Data Banks," CACM 13,6 (June, 1970), 377-387.

[DATE83]: Date, C.J. _An Introduction to Database Systems, vol. II_, Addison-Wesley, 1983.


APPENDIX A
THE PROGRAM

{$INCLUDE:'B:LOCGLBL.S.DOC'}
{$INCLUDE:'B:LOCTMCA.L.DOC'}
{$INCLUDE:'B:LOCPARMI.DOC'}

Program SimulateLocking(Input, Output, DetailFile, SummaryFile);

---------------
* PROGRAM TO SIMULATE VARIOUS LOCKING PROTOCOLS AND TO DETERMINE *
* THEIR EFFECTS ON EXECUTION DELAY IN A DATABASE SYSTEM. *
*
* WRITTEN BY Donald A. Varvel, August, 1984 *
* This program was written as part of the author's work on an *
* USAF-SCEEE grant June, 1984-August, 1984, and is based on ideas *
* developed by Donald A. Varvel and William Perrizo. *
*
* The files Input and Output are assumed to be interactive and *
* are used to obtain parameters. DetailFile receives a record of *
* each lock attempt and the processing of each lockable unit. It *
* should usually be WUL. SummaryFile is the main simulation *
* output, and should be CON (CRT screen) or PRN (the printer). *
*
* This version of this program is written for Microsoft Pascal *
* running under MS-DOS on a Zenith Z-100. Because of limitations *
* on disk space at compile time, it has been divided into a *
* PROGRAM and several UNITs and MODULEs. *
*
* OPERATION: *
* The main program sets some parameters and then generates a *
* number of simulated transactions on which to try the various *
* combinations of transaction managers and lock managers. *
* MAKETRANs generates transactions randomly within the given *
* parameters and MAKEDELAYS generates random processing delays. *
* These are the only random processes in the simulation. All of *
* the transaction managers are run against the same transactions *
* (by SIMTRANS), so the results for a given transaction are *
* strictly comparable. *
*
* As a standard of comparison, the procedure OPTIMAL has been *
* provided. It operates with more information than a real *
* transaction manager would have, and so does not represent a *
* practical implementation. *
---------------

Uses Globids, Tmcal, Parmi; (Unit Interfaces)

Var
I, J : Integer;
ID : 1..Algos;
Norm, Norm2 : Real;
Seed : Integer4;
P : T_L_Ptr;
InStr : Lstring(25); { Loop control, etc. }
{ Used in generating totals }
{ Normally-distributed random numbers }
{ Uniform random number seed }
{ Auxiliary pointer for Units }
{ Input string for overriding defaults }
Answer : Char;                  { Single-character input }
N_To_Sim : Integer;            { Number of transactions to simulate }

{********************}
Procedure Optimal(N_Units : Integer); Extern;
{********************}
Function Max(A, B : Integer) : Integer; Extern;
{********************}
Function RANDOM(var Seed : Integer4) : Real; Extern;
{********************}
Procedure NORMAL(Var Seed : Integer4; Var Result1, Result2 : Real); Extern;
{********************}
Procedure THO(LU_Num : Integer);
   { Simulates the usual blocking transaction }
   { manager with calls to lock manager 0.  }
Var
   I : Integer;
Begin
   Present_time := 0;              { Simulated clock }
   For I := 1 to LU_Num do begin
      Avail[I] := MaxInt;            { Initialization }
      Present_time := Present_time + LM0(I);  { Get lock }
      WriteIn(DetailFile, 'Look #', I:3, ' ', Present_time:6);
      Present_time := Present_time + Delay[I];  { Process }
      WriteIn(DetailFile, 'Process #', I:3, ' ', Present_time:6)
   End;
   Accumulate(Present_time, LU_Num, ID);  { For averages }
   Summary_Stats(FLOAT(Present_time), FLOAT(LU_Num))
End;

{********************}
Procedure TM1(Function Lock_Man(Loc_Num: Integer) : Boolean; LU_Num : Integer);
   { Transaction manager } from the paper:   }
   { While unprocessed units remain do begin   }
   {   Repeat Make nonblocking pass through list   }
   {   Until a pass acquires no locks;         }
   {   Issue a blocking request for some unit  }
   { End
Var
   I, LM_Calls : Integer;
   Flag : Boolean;
   Done : Booleanarray;
   Remaining : Integer;

Begin ( TM1 )                    { INITIALIZATIONS }
   ID := ID + 1;
   Remaining := LU_Num;
   LM_Calls := 0;
Present_time := 0;
For I := 1 to LU_Num do begin
  Done[I] := False;
  Avail[I] := Maxint
End;

{ NON-BLOCKING PASSES }
While Remaining > 0 do begin
  Repeat
    Flag := False; { Flag records recent lock acquisition }
    For I := 1 to LU_Num do
      If Not Done[I] then begin
        LM_Calls := LM_Calls + 1;
        If Lock_Man(I) then begin
          Present_time := Present_time + Lock_Request_Delay;
          Flag := True;
          Writeln(DetailFile, 'Lock #', I:3, ' ', Present_time:6);
          Present_time := Present_time + Delay[I];
          Writeln(DetailFile, 'Process #', I:3, ' ', Present_time:6);
          Remaining := Remaining - 1;
          Done[I] := True
        End    { Then part }
      End else begin
        Present_time := Present_time + Lock_Request_Delay;
        Writeln(DetailFile, 'Lock #', I:3, ',(u)', Present_time:6)
      End
    End    { If Not Done[I] ... }
  Until Not Flag;

  { WAIT FOR A LOCKABLE UNIT }
  { Find first not done }
  I := 1;
  While Done[I] and (I < LU_Num) do I := I + 1;
  { Blocking lock request }
  If Not Done[I] then begin
    Present_time := Present_time + LMO(I);
    LM_Calls := LM_Calls + 1;
    Remaining := Remaining - 1;
    Done[I] := True;
    Writeln(DetailFile, 'Lock #', I:3, '(w)', Present_time:6);
    Present_time := Present_time + Delay[I];
    Writeln(DetailFile, 'Process #', I:3, ', ', Present_time:6)
  End    { If Not Done[I] e.g., the blocking lock request }
End;    { While Remaining > 0 ... }

{ Summary stats }
Accumulate(Present_time, LM_Calls, ID);
Summary_stats(FLOAT(Present_time), FLOAT(LM_Calls)) End;

{ Procedure TH2 from the papers }
Procedure TH2(Function Lock_Man(Loc_Num: Integer): Boolean; LU_Num: Integer);
Make a nonblocking pass;

If unprocessed units remain then

Issue blocking request for a unit

End

Var

I, LM_Calls : Integer;
Done : Boolarray;
Remaining : Integer;

Begin

(INITIALIZATIONS)

ID := ID + 1;
Remaining := LU_Num;
LM_Calls := 0;
Present_time := 0;
For I := 1 to LU_Num do begin
Done[I] := False;
Avail[I] := Maxint
End;

While Remaining > 0 do begin

(Nonblocking pass)

For I := 1 to LU_Num do

If Not Done[I] then begin

LM_Calls := LM_Calls + 1;
If Lock_Man(I) then begin

Present_time := Present_time + Look_Request_Delay;
Writeln(DetailFile, 'Look @', I:3, Present_time:6);
Present_time := Present_time + Delay[I];
Writeln(DetailFile, 'Process @', I:3, Present_time:6);
Remaining := Remaining - 1;
Done[I] := True
End
(Then part)
Else begin

Present_time := Present_time + Look_Request_Delay;
Writeln(DetailFile, 'Look @', I:3, Present_time:6)
End
End;

(WAIT FOR A LOCKABLE UNIT)

(Find first not done)

I := 1;
While Done[I] and (I < LU_Num) do I := I + 1;
If Not Done[I] then begin

Present_time := Present_time + LMO(I);
LM_Calls := LM_Calls + 1;
Remaining := Remaining - 1;
Done[I] := True;
Writeln(DetailFile, 'Look @', I:3, '(N)', Present_time:6);
Present_time := Present_time + Delay[I];
Writeln(DetailFile, 'Process @', I:3, Present_time:6)
End
(The blocking look request)
End;

While Remaining > 0 do begin

I := 1;
While Done[I] and (I < LU_Num) do I := I + 1;
If Not Done[I] then begin

Present_time := Present_time + LMO(I);
LM_Calls := LM_Calls + 1;
Remaining := Remaining - 1;
Done[I] := True;
Writeln(DetailFile, 'Look @', I:3, '(N)', Present_time:6);
Present_time := Present_time + Delay[I];
Writeln(DetailFile, 'Process @', I:3, Present_time:6)
End
(The blocking look request)
End;
Accumulate(Present_time, LM_Calls, ID);
Summary_stats(Present_time, FLOAT(LM_Calls))
End;

************

Procedure TM3(Function Lock_Mem(Loc_Num: Integer): Boolean; LU_Num: Integer);
  { Transaction management from the paper: }
  { Process list using nonblocking requests: }
  { Process list using blocking requests }
Var
  I, LM_Calls : Integer;
  Done : Boolarray;
Begin
  { INITIALIZATIONS }
  ID := ID + 1;
  LM_Calls := 0;
  Present_time := 0;
  For I := 1 to LU_Num do begin
    Done[I] := False;
    Avail[I] := Maxint
  End;

  { NON-BLOCKING PASS }
  For I := 1 to LU_Num do begin
    LM_Calls := LM_Calls + 1;
    If Lock_Mem(I) then begin
      Present_time := Present_time + Lock_Request_Delay;
      Writeln(DetailFile, 'Lock ', I:3, ' ', Present_time:6);
      Present_time := Present_time + Delay[I];
      Writeln(DetailFile, 'Process ', I:3, ' ', Present_time:6);
      Done[I] := True
    End
    { Then part }
    Else begin
      Present_time := Present_time + Lock_Request_Delay;
      Writeln(DetailFile, 'Look ', I:3, ' (u) ', Present_time:6)
    End
  End;

  { BLOCKING PASS }
  For I := 1 to LU_Num do
    If Not Done[I] then begin
      LM_Calls := LM_Calls + 1;
      Present_time := Present_time + LMO(I);
      Writeln(DetailFile, 'Lock ', I:3, ' (w) ', Present_time:6);
      Present_time := Present_time + Delay[I];
      Writeln(DetailFile, 'Process ', I:3, ' ', Present_time:6)
    End;

  Accumulate(Present_time, LM_Calls, ID);
  Summary_stats(Present_time, FLOAT(LM_Calls))
End;
Procedure TM4(LU_No : Unit_Range); Extern;

Procedure Getanswer(Const S: String; Var Answer: Char); Extern;
    ( Get a 1-character response from keyboard )

Procedure Add_Links(Var List : T_L_Ptr; Start, Finish: Integer); Extern;

Procedure Terminate(Var List : T_L_Ptr); Extern;

Procedure MakeTrans(Var U : Un_Vec);
Var
    I, Tick, Lim, Duration : Integer;
Begin
    For I := 1 to LU_No do begin
        { Queue? }
        If RANDOM(Seed) <= Q_Prob then begin
            New(U[I]);
            NORMAL(Seed, Norm1, Norm2);
            Duration := TRUNC(FLOAT(Q.Std_Dev) * Norm1) + Q_Mean.Len;
            If Duration > 0 then begin
                New(U[I]);
                U[I].Time := Duration;
                U[I].Next := Nil
            End ( Then )
        End ( Else )
        Else begin
            U[I] := Nil;
            Tick := (-Look_Bar - 2 * Look_Sigma) DIV 10 * 10;
            Lim := -4 * Tick;
            While Tick < Lim do
                If RANDOM(Seed) > Ad_Req then Tick := Tick + 10
            Else begin
                NORMAL(Seed, Norm1, Norm2);
                Duration := TRUNC(FLOAT(Look_Sigma) * Norm1) + Look.Bar;
                If Duration <= 0 then Tick := Tick + 10
            Else begin
                Add_Links(U[I], Tick, Duration + Tick);
                Tick := (Tick + Duration) DIV 10 * 10 + 10
            End ( Else )
        End; ( Else and While )
        If U[I] <> Nil then Terminate(U[I])
    End ( Else )
End ( For )
End;

Procedure DispTrans; ( Display transactions )
Var P : T_L_Ptr;
    I : Integer;
Begin
    Writeln(SummaryFile); Writeln(SummaryFile);
    Writeln(SummaryFile, 'Transaction (\# = steady-state queue of given length)');
    Writeln(SummaryFile, 'Unit Delay Activity Activity Activity ...');
    For I := 1 to LU No do begin
        Write(SummaryFile, I:4, ' ', Delay[I]:3, ' ');
        P := Units[I];
        If P <> Nil then begin
            If P^.Next = Nil then Write(SummaryFile, ', P^.Time:8)
            Else Repeat
                P := P^.Next^.Next
                Until P^.Next = Nil
        End;
        Writeln(SummaryFile)
    End;
End;

[****************************************************************************************]
Procedure SimTrans;    { Simulate }
Var I : Integer;
Begin
    Writeln(SummaryFile);
    Write(SummaryFile, ' TM LM Time Active Lock Requests');
    Writeln(SummaryFile, ' Evaluation Eval/OptEval');
    Write(SummaryFile, ' Optimal ');
    Optimal(LU_No);
    Write(SummaryFile, ' 0 0 ');
    ID := 1;
    TM0(LU_No);
    Write(SummaryFile, ' 1 1 ');
    TM1(LM1, LU_No);
    Write(SummaryFile, ' 1 2 ');
    TM1(LM2, LU_No);
    Write(SummaryFile, ' 2 1 ');
    TM2(LM1, LU_No);
    Write(SummaryFile, ' 2 2 ');
    TM2(LM2, LU_No);
    Write(SummaryFile, ' 3 1 ');
    TM3(LM1, LU_No);
    Write(SummaryFile, ' 3 2 ');
    TM3(LM2, LU_No);
Write('Random number seed? ');
Readln(Seed);
Seed := Seed MOD 32768;  { Avoid overflow on first call to RANDOM }
Repeat  { Main program loop }
  For I := 0 to Algos do begin  { Clear totals }
    Totals[I].Time := 0;
    Totals[I].Requests := 0
  End;
  Repeat  { Minor input loop }
    Write('Enter number of lockable units per transaction: ');
    Readln(LU_No);
    If (LU_No < 1) or (LU_No > Max_Units) then
      Writeln('Must be in 0 < N <= ', Max_Units:1, '.
    Until (LU_No > 0) and (LU_No <= Max_Units);
  Repeat  { Minor input loop }
    Write('Enter number of transactions to simulate: ');
    Readln(N_To_Sim);
    If (N_To_Sim < 1) then Writeln('Must be positive.');
  Until (N_To_Sim > 0);  { Simulate N_To_Sim transactions }
  For I := 1 to N_To_Sim do begin
    MakeTrans(Units);
    MakeDelays(Delay, LU_No, Seed);
    DispsTrans;
SimTrans
End;
Averages(N_To_Sim);
Getanswer('Simulate another transaction? (Y/N) ', Answer)
Until (Answer <> 'Y') and (Answer <> 'y')
End.

{$INCLUDE:'B:LOCGLBS.DOC'}
{$INCLUDE:'B:LOCTMCAL.DOC'}

Module containing code to simulate the imaginary algorithm
OPTIMAL and the 4th modified transaction manager. Each
contains some internal procedures; notably, in the case of
TM4, the two parts of the third modified lock manager,
LM3I and LM3II.

Module Locopt;
Uses Globids, Tmcal;

Function Max(A, B : Integer) : Integer; Extern;

Procedure Optimal(LU Num : Integer);

Optimal determines a lower bound on processing the given
transaction using locks. By assumption it uses only as
many lock requests as there are lockable units and gets
into all queues at initiation time. It does a search of
the decision tree of orders of lock requests to find one
that results in the least delay. The treesearch selects
a first order of requests that is likely to be good, and
performs forward pruning according to two criteria; its
worst-case performance is O(N!), but is usually O(N).

Optimal contains the recursive treesearch Findbest, which
in turn contains Sort.

Type
Low Rec = Record
  When : Integer;
  Proc : 3..15
End;

Var
  Cutoff : Integer;  { Best time found so far. If it can't  }
                  { be underout, prune the present branch. }
  I : Integer;
  Remaining : Unit.Range;  { How many units remain to be processed? }
  Done : Boolean;
  Low_Vec : Array[0..Max_Units] of Low_Rec;  { Used in computing Lowerbound }

23
Function Findbest : Integer;
(*****************************************************************)
(* Recursive decision tree search, with cutoffs. *)
(*****************************************************************)

Type
ND_Rec = Record
    { Units not yet done and weights for sorting }
    U : Unit_Range;
    Val : Integer
End;
ND_Vec = Array[Unit_Range] of ND_Rec;

Var
Getlock, BestPath, Lowerbound, Pathtime : Integer;
Cursor : 0..Max_Units;
I : Unit_Range;
Notdone : ND_Vec;
P : T_L_Ptr;
J, Cum_Delay : Integer;

Procedure Sort(Var Tosort : ND_Vec; N : Unit_Range);
{*****************************************************************}
(* Linear insertion sort, in place. An O(N^2) sort *)
(* makes sense here, since it will be called far *)
(* more times with small N than with large. This *)
(* sort beats Shellsort and Quicksort for N less *)
(* than about 15, and N will seldom be that large. *)
{*****************************************************************}

Var
I, J, TempVal : Integer;
TempU : Unit_Range;

Begin
For I := 1 to N-1 do begin
    TempU := Tosort[I+1].U;
    TempVal := Tosort[I+1].Val;
    J := I;
    While TempVal < Tosort[J].Val do begin
        Tosort[J+1].U := Tosort[J].U;
        J := J - 1;
        If J < 1 then Break
    End; { While }
    Tosort[J+1].U := TempU;
    Tosort[J+1].Val := TempVal
End  { For I ... }

End;

{*****************************************************************}

Begin { Findbest }
If Remaining = 1 then begin
    { Only one unit remains }
    I := 1; While Done[I] do I := I + 1;
    Getlock := LNO(I) + Delay[I];
    If Present_time + Getlock < Cutoff then Cutoff := Present_time + Getlock;
    Findbest := Getlock
End;
End     { Else if Remaining = 1 ...  }
Else begin
    { More remaining one unit remains }
    Best_Path := Maxint;
    Lowerbound := 0;
    Cursor := 0;
    { Compute Lowerbound }
    For I := 1 to LU Num do
        If Not Done[I] then begin
            Cum_Delay := LMO(I);
            Low_VEC[0].When := Cum_Delay;
            Low_VEC[0].Proc := Delay[I];
            J := Cursor;
            While Low_VEC[J].When < Cum_Delay do begin
                Low_VEC[J+1].When := Low_VEC[J];
                J := J - 1
            end;  { While }
            Cursor := Cursor + 1;
            Low_VEC[J+1] := Low_VEC[0]
        end;  { Then and For }
    Cum_Delay := 0;
    For I := 1 to Cursor do begin
        Cum_Delay := Cum_Delay + Low_VEC[I].Proc;
        Lowerbound := Max(Lowerbound, Low_VEC[I].When + Cum_Delay);
        Cum_Delay := Cum_Delay + Lock_Request_Delay
    end;

    If Present_time + Lowerbound < Cutoff then begin
        { Arrange those units not processed }
        { Generate weights: Time of next looking for those units }
        { for those that are available and will not become }
        { are presently looked. }
        Cursor := 0;
        For I := 1 to LU Num do
            If Not Done[I] then begin
                Cursor := Cursor + 1;
                Notdone[Cursor].U := I;
                If Units[I].Next = Nil then Notdone[Cursor].Val := Delay[I] + 9000
                Else if (Units[I].Next = Nil) and (Units[I].Time <= Present_time) then Notdone[Cursor].Val := Delay[I] + 9000
                Else if Units[I].Next = Nil then Notdone[Cursor].Val := Units[I].Time + 10000
                Else begin
                    P := Units[I];
                    If P^.Next^.Next <> Nil then 
                        P := P^.Next^.Next;
                    If P^.Time > Present_time then Notdone[Cursor].Val := P^.Time
                        { ( PT ) }
                    Else if P^.Next^.Time > Present_time then
( () PT )
Else Notdone(Cursor).Val := Delay[I] + 9000
End;  ( Else )
End;  ( Then )

{ Sort according to weights }
Sort(Notdone, Cursor);

{ Search for optimal order, cutting off if equal to }
( a previously-computed lower bound or if unable to )
{ better the best previous time. }
I := 1;
While (I <= Cursor) and (Best Path > Lowerbound) do begin
Getlock := LMO(Notdone[I].U) + Delay[Notdone[I].U];
( Simulate processing the unit )
Done[Notdone[I].U] := True;
Present_time := Present_time + Getlock;
Remaining := Remaining - 1;
( Recurse )
Pathtime := Findbest;
( Record best found so far )
If Pathtime < Best Path - Getlock then
Best_Path := Pathtime + Getlock;
( Undo )
Done[Notdone[I].U] := False;
Present_time := Present_time - Getlock;
Remaining := Remaining + 1;
( Increment loop control )
I := I + 1
End;  ( While )
End;  ( Then )
Findbest := Best_Path
End;  ( Else )
End;  ( Findbest )

begin  ( Optimal )
( Optimal )
remaining := LU_num;
cutoff := Maxint;
present_time := 0;
for I := 1 to LU_num do begin
done[I] := False;
avail[I] := Maxint;
( Start all queues )
if units[I] <> Nil then if units[I]^.next = nil then
avail[I] := units[I]^.time
end;

( Call recursive triesearch )
I := findbest;
Opteval(float(cutoff), float(LU_num));  ( Record, for comparison )
Accumulate(Cutoff, LU_Num, 0);
Summary_Stats(FLOAT(Cutoff), FLOAT(LU_Num))
End;  "Optimal"

{******************************************************************************}
{******************************************************************************}
Procedure TM4(LU_Num : Unit_Range);
  [ Transaction manager 4: ]
  [ Make a nonblocking pass through list; ]
  [ While units remain do begin ]
  [ Wait for one to become available; ]
  [ Process it ]
  [ End ]

Type
  OR_Ptr = "OR_Rec;
  OR_Rec = Record
    LU : Unit_Range;
    Next : OR_Ptr
  End;

Var
  OR_List : OR_Ptr;
  LM_Calls, Remaining, I : Integer;
  Selected : Unit_Range;

{******************************************************************************}
{******************************************************************************}
Procedure AddtoOR(LU : Unit_Range; Var List : OR_Ptr);
  [ Add a lockable unit to this transaction's wait-list ]
Var P : OR_Ptr;
Begin
  New(P);
  P^.Next := List;
  P^.LU := LU;
  List := P
End;

{******************************************************************************}
{******************************************************************************}
Procedure DelOR(Var List : OR_Ptr; Val : Unit_Range);
  [ Delete a lockable unit from this transaction's wait-list ]
Var P : OR_Ptr;
Begin
  If List <> Nil then begin
    If List^.LU = Val then begin
      P := List;
      List := List^.Next;
      Dispose(P)
    End
    Else DelOR(List^.Next, Val)
  End
End;

{******************************************************************************}
Function LM3I(LU : Unit_Range) : Boolean;
  [ Look Manager 3, non-blocking part: ]
If unit is available then return true
Else begin
    Add unit to wait-list;
    Return false
End

Var
    T_L_Ptr;
Result : Boolean;
Begin
    If Units[LU] = Nil then Result := True
    Else if Units[LU].Next = Nil then begin
        Result := False;
        Avail[LU] := Present_time + Units[LU].Time
    End
    Else if Units[LU].Time > Present_time then Result := True
    Else begin
        P := Units[LU];
        Result := True;
        While P^.Time <= Present_time do begin
            If P^.Next^.Time > Present_time then begin
                Result := False;
            End;
            P := P^.Next^.Next
        End;
    End;
    If Not Result then AddtoOR(LU, OR_List);
LM3I := Result
End;

Function LM3W : Integer;
    { Lock Manager 3, blocking part: }
    { Set Selected to next unit available; }
    { Return delay for next unit }
Begin
    If OR_List = Nil then LM3W := -1   { Nothing left }
    Else begin
        { Finding next is not trivial! }
        Repeat
            LeastVal := Maxint;
            P := OR_List;
            While P^.Next <> Nil do begin
                If Avail(P^.LU) < LeastVal then begin
                    LeastVal := Avail(P^.LU);
                    Least := P^.LU
                End;
                P := P^.Next
            End;
LM3W := LeastVal
End

P : OR_Ptr;
UP : T_L_Ptr;
Least : Unit Range;
LeastVal, Waittime : Integer;
End;
If LeastVal < Present_time then
  If Units[Least]^ Next <> Nil then begin
    UP := Units[Least];
    While UP^.Time < Present_time do begin
      If UP^.Next^.Time > Present_time then
      UP := UP^.Next^.Next;
    end;
    Until Avail[Least] = LeastVal;
    DelOR(OR_List, Least);
    Selected := Least;
    LM3W := Max(Lock_Request_Delay, LeastVal - Present_time)
  end;
End;

{**********************************************************************}
Begin { Transaction Manager 4 itself }
{ Initializations }
Remaining := 0;
OR_List := Nil;
LM_Calls := 0;
Present_time := 0;

{ Non-blocking run }
For I := 1 to LU_Num do begin
  Avail[I] := Maxint;
  LM_Calls := LM_Calls + 1;
  If LM3I(I) then begin
    Present_time := Present_time + Lock_Request_Delay;
    Writeln(Detailfile, 'Lock #', I:3, 'T', Present_time:6);
    Present_time := Present_time + Delay[I];
    Writeln(Detailfile, 'Process #', I:3, 'T', Present_time:6)
  end;
Else begin
  Remaining := Remaining + 1;
  Present_time := Present_time + Lock_Request_Delay;
  Writeln(Detailfile, 'Lock #', I:3, 'T(u)', Present_time:6)
End;
End; { Non-blocking run }

{ Iteratively, wait for next available lock }
While Remaining > 0 do begin
  Present_time := Present_time + LM3W;
  Writeln(Detailfile, 'Lock #', Selected:3, '(o)', Present_time:6);
  Present_time := Present_time + Delay[Selected];
  Writeln(Detailfile, 'Process #', Selected:3, 'T', Present_time:6);
  Remaining := Remaining - 1;
  LM_Calls := LM_Calls + 1
End; { While = Wait for Locks }

Accumulate(Present_time, LM_Calls, R);
Summary_Stats(FLOAT(Present_time), FLOAT(LM_Calls))
End;

End.  { Module Locopt }

{=================================================================================================
  Interface that supplies global identifiers to all those *)
  program units that need them. Important consts and *)
  types are included, but also the files, lockable unit *)
  information Units, availability, processing delay, and *)
  the scalars Lock Request_Delay (presently 1), number of *)
  lockable units LU No, and Present_time. *)
{=================================================================================================}

Interface;
Unit Globids(Max_Units, Algos, Unit_Range, T_L_Ptr, Time_list,
  Un(Vec, Boolarray, Intarray, Detailfile, Summaryfile, Units,
  Lock_Request_delay, Present_time, Avail, Delay, LU_No);
Const
Max_Units = 100; { Max lookable units (arbitrary) }
Algos = 8; { Number of algorithms }
Type
Unit_Range = 1..Max_Units;
T_L_Ptr = "Time_list; { Time node for linked list }
Time_list = Record
  Time : Integer;
  Next : T_L_Ptr
End;
Un(Vec) = Array[Unit_Range] of T_L_Ptr;
{ Array of time lists }
Boolarray = Array[Unit_Range] of Boolean;
{ Type for Done, used in subprograms }
Intarray = Array[Unit_Range] of Integer;
{ Used for Avail and Delay }
Var
DetailFile, SummaryFile : Text; { Output files }
Units : Un(Vec); { Availability of lockable units:
  Always available }
Lock Request_Delay : Integer;  [ Time needed to access lock manager ]
Present_time : Integer;  [ Simulated clock ]
Avail, Delay : Intarray;  [ When available, Processing delay ]
LU_No : Integer;

Begin
End;

{$INCLUDE:'B:LOCGLBLS.DOc'}
Implementation of globids;
Begin
End.

{$INCLUDE:'B:LOCGLBLS.DOc'}
Module Helps;

[*****************************************************************************]
[* This module contains certain small, relatively pure *]
[* subprograms. They are declared EXTERN by those *]
[* calling units that access them. *]
[*****************************************************************************]

Uses Globids;

[*****************************************************************************]
Function Max(A, B : Integer) : Integer;
{ Returns value of maximum of two args }
Begin
  If A > B then Max := A
  Else Max := B
End;

[*****************************************************************************]
Function RANDOM(Var Seed : Integer4) : Real;
{ Generates uniform random floating-point numbers in the }
{ range 0 <= R < 1, using the linear congruence method }
{ of Knuth vol. 2. C is zero, so the low-order bits are }
{ not very random. Test high-order bits instead. }
Const Multiplier = 25997;
Modulus = 32768;
Fmod = 32768.0;
Begin
  Seed := (Seed * Multiplier) mod Modulus;
  RANDOM := FLOAT4(Seed) / Fmod
End;

[*****************************************************************************]
Procedure NORMAL(Var Seed : Integer4; Var Result1, Result2 : Real);
{ Generates two normally-distributed random numbers with }
{ a mean of zero and a standard deviation of one, using }
{ a method given in Knuth vol. 2. }

31
Var
   Root, V1, V2, S : Real;
Begin
   Repeat
      V1 := 2.0 * RANDOM(Seed) - 1;
      V2 := 2.0 * RANDOM(Seed) - 1;
      S := SQR(V1) + SQR(V2)
      Until (S < 1) and (S > 0);
      Root := SQRT(-2.0 * LN(S) / S);
      Result1 := V1 * Root;
      Result2 := V2 * Root
   Until (S < 1) and (S > 0);
End;

Procedure Getanswer(Constrs: String; Var Answer : Char);
   { Get a 1-character response from keyboard }
Begin
   Write(S);
   Readln(Answer)
End;

Procedure Add_Links(Var List : T_L_Ptr; Start, Finish : Integer);
   { Adds an activity entry to Units }
Begin
   If(Finish > 0) and (List = Nil) then begin
      New(List);
      List^.Time := Start;
      New(List^.Next);
      List^.Next^.Time := Finish;
      List^.Next^.Next := Nil
   End;
   Else if Finish > 0 then Add_Links(List^.Next^.Next, Start, Finish)
End;

Procedure Terminate(Var List : T_L_Ptr);
   { Terminates activity list with (Maxint, Nil) }
Begin
   If List = Nil then begin
      New(List);
      List^.Time := Maxint;
      List^.Next := Nil
   End;
   Else Terminate(List^.Next^.Next)
End;

Procedure MakeDelays(Var D : Intarray; LNo : Integer; Var Seed : Integer);
   { Generate random processing delays }
   { uniformly from 3 to 15. }
Begin
   For I := 1 to LNo do D[I] := TRUNC(13.0 * RANDOM(Seed)) + 3
End;

End.

{*******************************************************
 (* Simulation parameter initialization routines. *)
 {*******************************************************
Interface;
Unit Parmi(Override, Default, Display, Ad_Req, Lock_Bar, Lock_Sigma,
Q_Prob, Q_Mean.Len, Q_Sig.Dev);

Var Ad_Req : Real;    { Probability of an adverse request on }
                   { a lockable unit in a 10-tick period }
Lock_Bar : Integer;   { Mean adverse lock duration }
Lock_Sigma : Integer; { Std. deviation of adverse lock duration }
Q_Prob : Real;        { Probability of a queue on a unit }
Q_Mean.Len : Integer; { Mean queue length }
Q_Sig.Dev : Integer;  { Std. deviation of queue length }

Procedure Override;   { Override the default on one or more parameters }
Procedure Default;    { Set parameters to default }
Procedure Display(Var F: Text); { Display parameters }

End;

{$INCLUDE:'B:LOCPARMI.DOC'}

{*******************************************************
 (* Implementation of parameter Init & display routines. *)
 {*******************************************************
Implementation of Parmi;

{*******************************************************
 (* Parameters. These are the defaults. *)
 {*******************************************************
Const Con_Def = 0.01;    { Default probability of adverse lock request/10 ticks }
Lock_Default = 150;      { Default mean lock duration }
Lk_Sig_Default = 50;     { Default standard deviation of lock duration }
Q_Prob_Default = 0.1;    { Default probability of queue }
Q_Bar = 100;             { Default mean queue length }
Q_Sig = 20;              { Default queue std. dev. }

Procedure Override;   { Override the default on one or more parameters }
Var
InStr : Lstring(25);

Begin
  Writeln('Enter number (or <ENTER> for default).');
  Write('Prob. of adverse request per 10 ticks (', Ad_Rq:6:4, ')');
  Readln(InStr);
  If (InStr<>Null)then if Not DECODE(InStr, Ad_Rq) then Ad_Rq := Con_Def;
  Readln(InStr);
  If (InStr <> Null) then if Not DECODE(InStr, Look_Bar) then
    Lock_Bar := Lock_Default;
  Write('Std. devition of look duration ('', Look_Sigma:1, ')');
  Readln(InStr);
  If (InStr <> Null) then if Not DECODE(InStr, Look_Sigma) then
    Look_Sigma := Lk_Sig_Default;
  Write('Per-unit probability of a queue ('', Q_Prob:6:4, ')');
  Readln(InStr);
  If (InStr <> Null) then if Not DECODE(InStr, Q_Prob) then
    Q_Prob := Q_Prob_Default;
  Write('Probable length of queue in ticks ('', Q_Mean.Len:1, ')');
  Readln(InStr);
  If (InStr <> Null) then if Not DECODE(InStr, Q_Mean.Len) then
    Q_Mean.Len := Q_Bar;
  Write('Std. deviation of queue length ('', Q.Std.Dev:1, ')');
  Readln(InStr);
  If (InStr <> Null) then if Not DECODE(InStr, Q.Std.Dev) then
    Q.Std.Dev := Q_Sigma
End;

{ Set parameters to default }

Procedure Default;
  Begin
    Ad_Rq := Con_Def;
    Look_Bar := Lock_Default;
    Look_Sigma := Lk_Sig_Default;
    Q_Prob := Q_Prob_Default;
    Q_Mean.Len := Q_Bar;
    Q.Std.Dev := Q_Sigma
  End;

{ Display parameters to file F }

Procedure Display;
  Begin
    Writeln(F, 'Parameters Probability Mean Std. Dev.',); 
    Writeln(F, '-----------------------------------------'); 
    Writeln(F, 'Potential conflicts', Ad_Rq:10:3, Look_Bar:10, 
      Look_Sigma:8); 
    Writeln(F, 'Queues', Q_Prob:10:3, Q_Mean.Len:10, 
      Q.Std.Dev:8); 
    Writeln(F)
  End;

34
End.

{*******************************************************************************}
{ * Interface for routines called by transaction managers * }
{*******************************************************************************}

Interface;

Unit Tmcal(TotRec, Totals, LMO, LM1, LM2, Summary_stats, Accumulate,
    Averages, Opteval);

Const Algos = 8;

Type TotRec = Record Time, Requests : Integer4 End;

Var Totals : Array[0..Algos] of TotRec;

Function LMO(LU : Integer) : Integer;
    { The usual blocking lock manager, with the }  
    { addition of a check to Avail to go with }  
    { lock manager 2. }

Function LM1(LU : Integer) : Boolean;

Function LM2(LU : Integer) : Boolean;

Procedure Summary_Stats(Time, Requests : Real);

Procedure Accumulate(Time, Requests, ID : Integer);
    { Accumulate totals for a particular trans. type }

Procedure Averages(TNum : Integer);

Procedure Opteval(Time, Requests : Real);

End;

{$INCLUDE:'B:LOCTMCAL.DOC'}
{$INCLUDE:'B:LOCGLBLS.DOC'}

{*******************************************************************************}
{ * Implementation of Lock Managers 0-2, Summary_stats, Accumulate, }  
{ * Averages, and Opteval. * }
{*******************************************************************************}

Implementation of Tmcal;
Uses Globids;

Var BestE : Real;    { Evaluation of OPTIMAL, for comparison }

Function Max(A, B : Integer) : Integer; Extern;

Function LNU;
    { The usual blocking lock manager, with the addition of }  
    { a check to Avail to go with lock manager 2. }  
Var P : T_L_Ptr;

35
Begin
If Units[LU] = Nil Then LMO := Lock_Request_Delay
Else if Avail[LU] <> Maxint Then
   LMO := Max(Avail[LU] - Present_time, Lock_Request_Delay)
Else if Units[LU]^'.Next = Nil Then
   LMO := Max(Lock_Request_Delay, Units[LU]^'.Time)
Else if Units[LU]^'.Time > Present_Time Then
   LMO := Lock_Request_Delay
Else begin
   P := Units[LU];
   LMO := Lock_Request_Delay;
   While P^'.Time <= Present_time do begin
      If P^.Next^'.Time > Present_time then
         LMO := Max(P^.Next^'.Time - Present_time, Lock_Request_delay);
         P := P^.Next^'.Next
   End
End
End;

{FUNCTION L1
(A simulator of Lock Manager 1: This is the nonblocking part.)
(LMO is used for the blocking part. This lock manager simply)
(determines whether the requested lock is available and)
 RETURNS TRUE OR FALSE.
Var P : T_L_Ptr;
Begin
   If Units[LU] = Nil then L1 := True    [No activity]
   Else if Units[LU]^'.Next = Nil Then L1 := False    [Queue]
   Else if Units[LU]^'.Time > Present_time Then L1 := True    [Not yet started]
   Else begin [Do an activity's start and finish bracket Present_time?]
      P := Units[LU];
      L1 := True;
      While P^'.Time <= Present_time do begin
         If P^.Next^'.Time > Present_time then L1 := False;
         P := P^.Next^'.Next
      End
   End
End;

{FUNCTION L2
(Simulator of Lock Manager 2:)
(This is the nonblocking part only. LMO is used for the)
(blocking part. This lock manager simulates placing the)
(transaction in the queue if the unit is not immediately)
(available)
Var P : T_L_Ptr;
Begin
   If Units[LU] = Nil then L2 := True    [No activity]
   Else if Avail[LU] <> Present_time then L2 := True    [Presently available]
   Else if Avail[LU] <> Maxint then L2 := False    [Not available yet]
   Else if Units[LU]^'.Next = Nil then begin    [Queue]
      Avail[LU] := Present_time + Units[LU]^'.Time;
   End
End;

}
LM2 := False
Else if Units[LJ].Time > Present_time then LM2 := True  { Future activity }
Else begin  { Do an activity's start and finish bracket Present_time? }
    P := Units[LJ];
    LM2 := True;
    While P^.Time <= Present_time do begin
        If P^.Next^.Time > Present_time then begin
            LM2 := False;
        End;
        P := P^.Next^.Next
    End     { While P^.Time ... }
End     { Else ... }
End;

Procedure Summary_Stats;
Var Eval : Real;
Begin
    Write(SummaryFile, Time: 11:1);
    Write(SummaryFile, Requests: 12:1);
    Eval := (1.0 + SQR(1.0 - Requests/LU_No)) * Time;
    Write(SummaryFile, Eval:15:2);
    Writeln(SummaryFile, Eval / BestE :15:3)
End;

Procedure Accumulate;
  { Accumulate totals for a particular trans. type } 
Begin
    Totals[ID].Time := Totals[ID].Time + Time;
    Totals[ID].Requests := Totals[ID].Requests + Requests
End;

Procedure Averages;
  { Display averages for a particular trans. type }
Var I : Integer;
Begin
    Opteval(Totals[0].Time / T_Num, Totals[0].Requests / T_num);
    Write(SummaryFile); Write(SummaryFile, '%%%%%%%%%%%%%%%%%');
    Write(SummaryFile, 'Averages for this transaction type:');
    Write(SummaryFile, ' TM LM Time Active Look Requests');
    Write(SummaryFile, ' Evaluation Eval/OptEval');
    Write(SummaryFile, ' Optimal ');
    Summary_stats(Totals[0].Time / T_Num, Totals[0].Requests / T_Num);
    Write(SummaryFile, ' 0 0 ');
    Summary_stats(Totals[1].Time / T_Num, Totals[1].Requests / T_Num);
    For I := 2 to Algos-1 do begin
        Write(SummaryFile, I DIV 2 : 4, I MOD 2 + 1 : 6, ' ');
        Summary_stats(Totals[I].Time / T_Num, Totals[I].Requests / T_Num)
    End;
    Write(SummaryFile, 4:4, 3:6, '');
Summary_stats(Totals[8].Time / T_num, Totals[8].Requests / T_Num)
End;

Procedure OptEval;
{ Record evaluation of OPTIMAL for comparison with others }
Begin
  BestE := (1.0 + SQR(1.0 - Requests / LU_No)) * Time
End;
End.
APPENDIX B: RESULTS

LIGHT ACTIVITY

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Probability</th>
<th>Mean</th>
<th>Std. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Potential conflicts</td>
<td>0.004</td>
<td>150</td>
<td>50</td>
</tr>
<tr>
<td>Queues</td>
<td>0.080</td>
<td>250</td>
<td>80</td>
</tr>
</tbody>
</table>

Averages for 5 lockable units:

<table>
<thead>
<tr>
<th>TM</th>
<th>LM</th>
<th>Time Active</th>
<th>Lock Requests</th>
<th>Evaluation</th>
<th>Eval/OptEval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>136.1</td>
<td>5.0</td>
<td>136.05</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>162.4</td>
<td>5.0</td>
<td>162.45</td>
<td>1.194</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>160.5</td>
<td>6.2</td>
<td>168.99</td>
<td>1.242</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>145.1</td>
<td>6.2</td>
<td>152.72</td>
<td>1.123</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>160.1</td>
<td>5.6</td>
<td>162.41</td>
<td>1.194</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>145.1</td>
<td>5.6</td>
<td>147.14</td>
<td>1.082</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>160.1</td>
<td>5.6</td>
<td>161.99</td>
<td>1.191</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>145.1</td>
<td>5.6</td>
<td>146.81</td>
<td>1.079</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>144.4</td>
<td>5.6</td>
<td>146.15</td>
<td>1.074</td>
</tr>
</tbody>
</table>

Averages for 10 lockable units:

<table>
<thead>
<tr>
<th>TM</th>
<th>LM</th>
<th>Time Active</th>
<th>Lock Requests</th>
<th>Evaluation</th>
<th>Eval/OptEval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>230.0</td>
<td>10.0</td>
<td>230.00</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>376.2</td>
<td>10.0</td>
<td>376.20</td>
<td>1.636</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>353.5</td>
<td>13.6</td>
<td>400.65</td>
<td>1.742</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>261.0</td>
<td>13.5</td>
<td>292.92</td>
<td>1.274</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>352.5</td>
<td>12.4</td>
<td>372.02</td>
<td>1.617</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>261.0</td>
<td>12.3</td>
<td>274.16</td>
<td>1.192</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>351.9</td>
<td>11.6</td>
<td>361.43</td>
<td>1.571</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>261.0</td>
<td>11.6</td>
<td>268.05</td>
<td>1.165</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>259.0</td>
<td>11.6</td>
<td>266.00</td>
<td>1.157</td>
</tr>
</tbody>
</table>

Averages for 15 lockable units:

<table>
<thead>
<tr>
<th>TM</th>
<th>LM</th>
<th>Time Active</th>
<th>Lock Requests</th>
<th>Evaluation</th>
<th>Eval/OptEval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>244.6</td>
<td>15.0</td>
<td>244.55</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>476.3</td>
<td>15.0</td>
<td>476.25</td>
<td>1.947</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>452.5</td>
<td>20.0</td>
<td>501.72</td>
<td>2.052</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>300.4</td>
<td>19.6</td>
<td>329.27</td>
<td>1.346</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>450.9</td>
<td>18.1</td>
<td>470.11</td>
<td>1.922</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>300.9</td>
<td>18.0</td>
<td>312.49</td>
<td>1.278</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>449.9</td>
<td>17.0</td>
<td>457.90</td>
<td>1.872</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>300.9</td>
<td>17.0</td>
<td>306.20</td>
<td>1.252</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>298.3</td>
<td>17.0</td>
<td>303.55</td>
<td>1.241</td>
</tr>
</tbody>
</table>

Averages for 20 lockable units:

<table>
<thead>
<tr>
<th>TM</th>
<th>LM</th>
<th>Time Active</th>
<th>Lock Requests</th>
<th>Evaluation</th>
<th>Eval/OptEval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>270.3</td>
<td>20.0</td>
<td>270.25</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>718.4</td>
<td>20.0</td>
<td>718.40</td>
<td>2.658</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>651.3</td>
<td>27.1</td>
<td>733.32</td>
<td>2.714</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>354.1</td>
<td>26.7</td>
<td>393.84</td>
<td>1.457</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>655.2</td>
<td>24.5</td>
<td>688.37</td>
<td>2.547</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>354.4</td>
<td>24.3</td>
<td>370.35</td>
<td>1.370</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>653.6</td>
<td>22.6</td>
<td>664.65</td>
<td>2.459</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>355.5</td>
<td>22.6</td>
<td>361.46</td>
<td>1.337</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>352.0</td>
<td>22.6</td>
<td>357.95</td>
<td>1.385</td>
</tr>
</tbody>
</table>
## MEDIUM ACTIVITY

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Probability</th>
<th>Mean</th>
<th>Std. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Potential conflicts</td>
<td>0.010</td>
<td>150</td>
<td>50</td>
</tr>
<tr>
<td>Queues</td>
<td>0.100</td>
<td>250</td>
<td>80</td>
</tr>
</tbody>
</table>

### Averages for 5 lockable units:

<table>
<thead>
<tr>
<th>TM</th>
<th>LM</th>
<th>Time</th>
<th>Active</th>
<th>Look Requests</th>
<th>Evaluation</th>
<th>Eval/OptEval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td></td>
<td>183.5</td>
<td>5.0</td>
<td>183.50</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td>259.0</td>
<td>5.0</td>
<td>259.00</td>
<td>1.411</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>254.0</td>
<td>6.3</td>
<td>361.32</td>
<td>1.468</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 2</td>
<td>192.9</td>
<td>8.1</td>
<td>264.61</td>
<td>1.442</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 1</td>
<td>253.4</td>
<td>6.9</td>
<td>291.88</td>
<td>1.591</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 2</td>
<td>192.9</td>
<td>6.8</td>
<td>217.84</td>
<td>1.187</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 1</td>
<td>252.8</td>
<td>6.3</td>
<td>269.84</td>
<td>1.470</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 2</td>
<td>192.9</td>
<td>6.3</td>
<td>205.89</td>
<td>1.122</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 3</td>
<td>190.8</td>
<td>6.3</td>
<td>203.70</td>
<td>1.110</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Averages for 10 lockable units:

<table>
<thead>
<tr>
<th>TM</th>
<th>LM</th>
<th>Time</th>
<th>Active</th>
<th>Look Requests</th>
<th>Evaluation</th>
<th>Eval/OptEval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td></td>
<td>207.6</td>
<td>10.0</td>
<td>207.55</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td>427.3</td>
<td>10.0</td>
<td>427.25</td>
<td>2.409</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>382.5</td>
<td>15.5</td>
<td>498.21</td>
<td>2.090</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 2</td>
<td>243.2</td>
<td>15.1</td>
<td>305.22</td>
<td>1.471</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 1</td>
<td>381.0</td>
<td>13.1</td>
<td>418.86</td>
<td>2.018</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 2</td>
<td>244.9</td>
<td>12.8</td>
<td>263.42</td>
<td>1.269</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 1</td>
<td>381.0</td>
<td>12.3</td>
<td>401.10</td>
<td>1.933</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 2</td>
<td>245.9</td>
<td>12.3</td>
<td>258.91</td>
<td>1.247</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 3</td>
<td>236.1</td>
<td>12.3</td>
<td>248.54</td>
<td>1.197</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Averages for 15 lockable units:

<table>
<thead>
<tr>
<th>TM</th>
<th>LM</th>
<th>Time</th>
<th>Active</th>
<th>Look Requests</th>
<th>Evaluation</th>
<th>Eval/OptEval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td></td>
<td>257.1</td>
<td>15.0</td>
<td>257.10</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td>586.2</td>
<td>15.0</td>
<td>586.20</td>
<td>2.280</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>486.4</td>
<td>22.3</td>
<td>599.97</td>
<td>2.334</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 2</td>
<td>303.1</td>
<td>21.7</td>
<td>363.57</td>
<td>1.414</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 1</td>
<td>484.0</td>
<td>19.3</td>
<td>522.91</td>
<td>2.034</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 2</td>
<td>305.0</td>
<td>18.9</td>
<td>325.04</td>
<td>1.264</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 1</td>
<td>491.0</td>
<td>17.6</td>
<td>506.32</td>
<td>1.969</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 2</td>
<td>306.3</td>
<td>17.6</td>
<td>315.81</td>
<td>1.228</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 3</td>
<td>298.0</td>
<td>17.6</td>
<td>307.30</td>
<td>1.195</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Averages for 20 lockable units:

<table>
<thead>
<tr>
<th>TM</th>
<th>LM</th>
<th>Time</th>
<th>Active</th>
<th>Look Requests</th>
<th>Evaluation</th>
<th>Eval/OptEval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td></td>
<td>299.0</td>
<td>20.0</td>
<td>299.00</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td>845.0</td>
<td>20.0</td>
<td>845.05</td>
<td>2.826</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>663.0</td>
<td>29.7</td>
<td>819.02</td>
<td>2.739</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 2</td>
<td>383.8</td>
<td>28.4</td>
<td>451.50</td>
<td>1.510</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 1</td>
<td>660.8</td>
<td>26.3</td>
<td>725.39</td>
<td>2.426</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 2</td>
<td>385.5</td>
<td>25.5</td>
<td>415.13</td>
<td>1.388</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 1</td>
<td>661.2</td>
<td>24.0</td>
<td>684.99</td>
<td>2.298</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 2</td>
<td>390.4</td>
<td>24.0</td>
<td>405.58</td>
<td>1.356</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 3</td>
<td>380.5</td>
<td>24.0</td>
<td>395.29</td>
<td>1.328</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Heavy Activity

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Probability</th>
<th>Mean</th>
<th>Std. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Potential conflicts</td>
<td>0.050</td>
<td>150</td>
<td>50</td>
</tr>
<tr>
<td>Queues</td>
<td>0.150</td>
<td>250</td>
<td>80</td>
</tr>
</tbody>
</table>

### Averages for 5 Lockable Units:

<table>
<thead>
<tr>
<th>TM</th>
<th>LM</th>
<th>Time Active</th>
<th>Lock Requests</th>
<th>Evaluation</th>
<th>Eval/OptEval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>201.6</td>
<td>5.0</td>
<td>201.55</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td>407.4</td>
<td>5.0</td>
<td>407.40</td>
<td>2.021</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>328.4</td>
<td>11.2</td>
<td>833.22</td>
<td>4.134</td>
<td></td>
</tr>
<tr>
<td>1 2</td>
<td>222.1</td>
<td>10.8</td>
<td>515.94</td>
<td>2.560</td>
<td></td>
</tr>
<tr>
<td>2 1</td>
<td>328.0</td>
<td>8.7</td>
<td>507.54</td>
<td>2.518</td>
<td></td>
</tr>
<tr>
<td>2 2</td>
<td>222.1</td>
<td>8.1</td>
<td>310.32</td>
<td>1.540</td>
<td></td>
</tr>
<tr>
<td>3 1</td>
<td>334.7</td>
<td>7.3</td>
<td>405.52</td>
<td>2.012</td>
<td></td>
</tr>
<tr>
<td>3 2</td>
<td>223.4</td>
<td>7.3</td>
<td>270.73</td>
<td>1.343</td>
<td></td>
</tr>
<tr>
<td>4 3</td>
<td>218.3</td>
<td>7.3</td>
<td>264.43</td>
<td>1.312</td>
<td></td>
</tr>
</tbody>
</table>

### Averages for 10 Lockable Units:

<table>
<thead>
<tr>
<th>TM</th>
<th>LM</th>
<th>Time Active</th>
<th>Lock Requests</th>
<th>Evaluation</th>
<th>Eval/OptEval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>278.1</td>
<td>10.0</td>
<td>278.10</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td>732.5</td>
<td>10.0</td>
<td>732.45</td>
<td>2.634</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>528.0</td>
<td>26.2</td>
<td>1913.50</td>
<td>6.881</td>
<td></td>
</tr>
<tr>
<td>1 2</td>
<td>320.1</td>
<td>23.5</td>
<td>903.62</td>
<td>3.249</td>
<td></td>
</tr>
<tr>
<td>2 1</td>
<td>529.5</td>
<td>19.8</td>
<td>1037.93</td>
<td>3.732</td>
<td></td>
</tr>
<tr>
<td>2 2</td>
<td>320.1</td>
<td>18.0</td>
<td>527.62</td>
<td>1.897</td>
<td></td>
</tr>
<tr>
<td>3 1</td>
<td>601.7</td>
<td>15.3</td>
<td>767.54</td>
<td>2.760</td>
<td></td>
</tr>
<tr>
<td>3 2</td>
<td>326.8</td>
<td>15.3</td>
<td>416.87</td>
<td>1.499</td>
<td></td>
</tr>
<tr>
<td>4 3</td>
<td>307.1</td>
<td>15.3</td>
<td>391.74</td>
<td>1.409</td>
<td></td>
</tr>
</tbody>
</table>

### Averages for 15 Lockable Units:

<table>
<thead>
<tr>
<th>TM</th>
<th>LM</th>
<th>Time Active</th>
<th>Lock Requests</th>
<th>Evaluation</th>
<th>Eval/OptEval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>290.0</td>
<td>15.0</td>
<td>289.95</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td>1097.6</td>
<td>15.0</td>
<td>1097.60</td>
<td>3.785</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>656.8</td>
<td>45.0</td>
<td>3284.00</td>
<td>11.326</td>
<td></td>
</tr>
<tr>
<td>1 2</td>
<td>342.3</td>
<td>42.6</td>
<td>1500.97</td>
<td>5.177</td>
<td></td>
</tr>
<tr>
<td>2 1</td>
<td>696.3</td>
<td>33.3</td>
<td>1732.80</td>
<td>5.976</td>
<td></td>
</tr>
<tr>
<td>2 2</td>
<td>352.5</td>
<td>30.0</td>
<td>702.75</td>
<td>2.424</td>
<td></td>
</tr>
<tr>
<td>3 1</td>
<td>798.5</td>
<td>23.5</td>
<td>1051.97</td>
<td>3.628</td>
<td></td>
</tr>
<tr>
<td>3 2</td>
<td>367.6</td>
<td>23.5</td>
<td>484.26</td>
<td>1.670</td>
<td></td>
</tr>
<tr>
<td>4 3</td>
<td>330.6</td>
<td>23.5</td>
<td>435.58</td>
<td>1.502</td>
<td></td>
</tr>
</tbody>
</table>

### Averages for 20 Lockable Units:

<table>
<thead>
<tr>
<th>TM</th>
<th>LM</th>
<th>Time Active</th>
<th>Lock Requests</th>
<th>Evaluation</th>
<th>Eval/OptEval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>328.8</td>
<td>20.0</td>
<td>328.75</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td>1381.4</td>
<td>20.0</td>
<td>1381.40</td>
<td>4.202</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>977.3</td>
<td>57.3</td>
<td>4376.79</td>
<td>13.313</td>
<td></td>
</tr>
<tr>
<td>1 2</td>
<td>391.6</td>
<td>46.8</td>
<td>1097.38</td>
<td>3.338</td>
<td></td>
</tr>
<tr>
<td>2 1</td>
<td>980.5</td>
<td>41.8</td>
<td>2150.89</td>
<td>6.543</td>
<td></td>
</tr>
<tr>
<td>2 2</td>
<td>397.9</td>
<td>36.3</td>
<td>662.20</td>
<td>2.014</td>
<td></td>
</tr>
<tr>
<td>3 1</td>
<td>1154.2</td>
<td>30.4</td>
<td>1463.30</td>
<td>4.451</td>
<td></td>
</tr>
<tr>
<td>3 2</td>
<td>418.3</td>
<td>30.4</td>
<td>530.26</td>
<td>1.613</td>
<td></td>
</tr>
<tr>
<td>4 3</td>
<td>394.5</td>
<td>30.4</td>
<td>500.21</td>
<td>1.522</td>
<td></td>
</tr>
</tbody>
</table>
DISTRIBUTION LIST

HQ USAF/MDORC
Washington, D.C.  20330  (1)

Air Force Systems Command
Andrews AFB, MD 20332
AFSC/VR   (1)
AFSC/XRK   (2)

Electronic Systems Division
Hanscom AFB, MA  01731
AFGL/SULR  (3)
AFGL/SULL  (1)
ESD/CC   (1)
ESD/DC   (1)
ESD/IN   (1)
ESD/OC   (1)
ESD/TC   (2)
ESD/TO   (1)
ESD/YW   (1)
ESD/XR   (2)
ESD/XRC  (2)
ESD/XRK  (2)
ESD/XRT  (10)
ESD/XRW  (1)

ESD DET 9
APO NY 09021  (2)

Rome Air Development Center
Griffiss AFB, NY  13441
RADC/CC   (1)
RADC/CA   (2)
RADC/DC   (2)
RADC/CO   (2)
RADC/OC   (2)
RADC/XP   (2)

HQ ESC/XDX
Kelly AFB
San Antonio, TX  78243  (2)

Tactical Air Command
Langley AFB, VA 23665
TAFITG/II  (2)
TAC/DRC  (2)

Tactical Air Command Systems Office
Hanscom AFB, MA  01731
TACSO-E  (2)

The MITRE Corporation
Bedford Operations
P.O. Box 208
Bedford, MA  01730
ATTN: Mr Norm Briggs  (2)

DARPA/ITPO
1400 Wilson Blvd
Arlington, VA  22209  (1)

USA/CECOM
Pt. Monmouth, NJ  07703  (1)

Defense Technical Information Center
Cameron Station
Alexandria, VA  22314  (2)

AU Library
Maxwell AFB, AL  36112  (1)