TPC Benchmark B
Full Disclosure Report
for the
DECsystem 5000 Model 25
Using
ULTRIX 4.2A and INFORMIX-OnLine 4.10

Company Name	System Name	Database Software	Operating System Software
Digital Equipment Corporation	**DECsystem 5000 Model 25**	**INFORMIX-OnLine 4.10**	**ULTRIX 4.2A**

Total System Cost	TPC-B Throughput	Price Performance
-Hardware -Software -5 years Maintenance	Sustained maximum throughput of system running TPC Benchmark B expressed in transactions per second.	Total system cost/ TPC-B throughput ($65,096/23.8 tpsB)
$65,096	23.8 tpsB	$2,735 per tpsB

TM

Submitted for Review: 4/23/92

First Printing April 1992

Digital Equipment Corporation believes that the information in this document is accurate as of its publication date. The information in this document is subject to change without notice. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The pricing information in this document accurately reflects prices in effect on the indicated dates. However, Digital Equipment Corporation provides no warranty on the pricing information in this document.

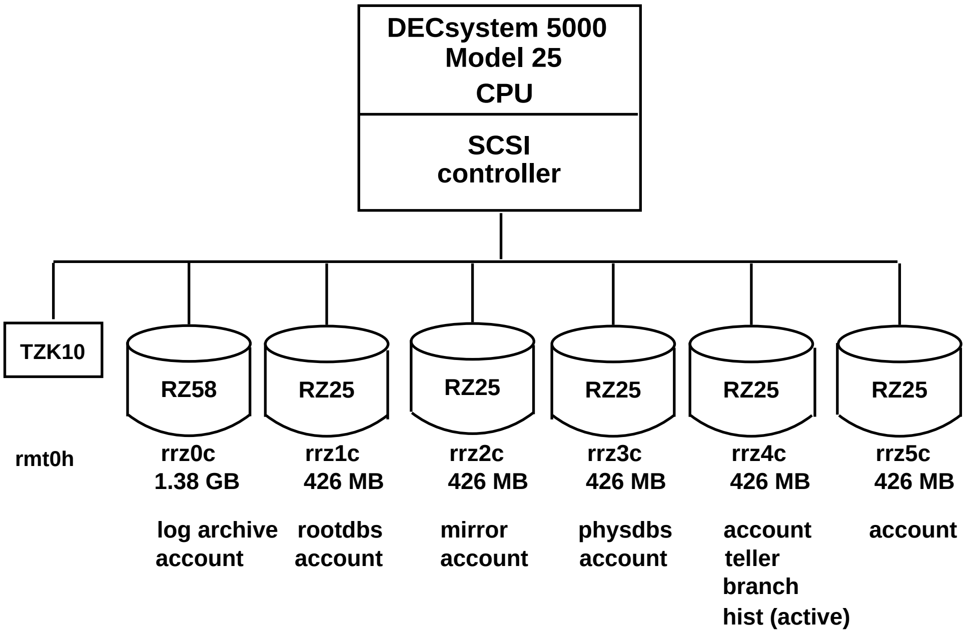
The performance information in this document is for guidance only. System performance is highly dependent on many factors including system hardware, system and user software, and user application characteristics. Customer applications must be carefully evaluated before estimating performance. Digital Equipment Corporation does not warrant or represent that a user can or will achieve similar performance expressed in transactions per second (TPS) or normalized price/performance (K$/TPS). No warranty on system performance or price/performance is expressed or implied in this document.

Digital Equipment Corporation	DECsystem 5000 Model 25	TPC-B Rev. 1.1 Report Date: 23 Apr 1992

Total System Cost	TPC-B Throughput	Price/Performance
$65,096	23.8 tpsB	$2,735 per tpsB

Processors	Database Manager	Operating System	Other Software	Concurrency Level
1 DECsystem 5000 Model 25	INFORMIX-OnLine V4.10 (Relational)	ULTRIX 4.2A	INFORMIX-ESQL/C V4.10	9

Priced System Configuration



System Components	Qty	Description
Processors	1	DECsystem 5000 Model 25
Memory		40 MB of main memory
Disk Controllers	1	SCSI controller
Disk Drives	1	1.38 GB RZ58 SCSI Disk
	5	426 MB RZ25 SCSI Disks
Total GB of Disk Storage		3.51 GB
Tapes Drive	1	TLZ04 525 MB Quarter-Inch Cartridge Tape Drive

Digital Equipment Corporation	DECsystem 5000 Model 25	TPC-B Rev. 1.1 Report Date: 23 Apr 1992

DESCRIPTION	MODEL #	UNIT PRICE*	QTY	EXTENDED PRICE*	YRS. 2-5 MAINTEN.	TOTAL 5 YR COST
Host and Database						
Hardware (24 MARCH 1992):						
DECsystem 5000 Model 25, 24 MB	PM319-RX	$12,659.00	1	$12,659.00	$0.00	$12,659.00
Warranty Upgrade to DS9	FM-DECUP-12	$132.00	1	$132.00	$8,304.00	$8,436.00
Additional 16 MB Memory	MS01-CA	$3,200.00	1	$3,200.00	$0.00	$3,200.00
2 RZ25 426 MB Disks	SZ12G-GA	$5,600.00	2	$11,200.00	$4,800.00	$16,000.00
1 1.38 GB RZ58 Disk, TLZ04	SZ12J-EA	$9,284.00	1	$9,284.00	$3,936.00	$13,220.00
Software:						
ULTRIX	ULTRIX 4.2	$0.00	1	$0.00	$0.00	$0.00
UX-32 Media & Doc.	QA-VYVAA-H5	$1,315.00	1	$1,315.00	$0.00	$1,315.00
Digital SUB TOTAL				$37,790.00	$17,040.00	$54,830.00
Years 2-5 Warranty Adder = 5.3725%					$915.47	$915.47
Digital SUB TOTAL				$37,790.00	$17,955.47	$55,745.47
Prepayment Maintenance Discount = 25%					($4,488,87)	($4,488.87)
Digital TOTAL				$37,790.00	$13,466.60	$51,256.60
Informix						
INFORMIX Software (Class "D" License) (1 AUGUST 1991):						
INFORMIX-OnLine 4.10 (16U)	Full Dev./Run T	$6,700.00	1	$6,700.00	$4,840.00	$11,540.00
INFORMIX-ESQL/C 4.10 (16U)	Full Dev./Run T	$1,340.00	1	$1,340.00	$960.00	$2,300.00
Informix TOTAL				$8,040.00	$5,800.00	$13,840.00
CONFIGURATION TOTALS				$45,830.00	$19,266.60	$65,096.60
tpsB & $/tpsB					23.8	$2,735

*Includes 1 year warranty

Abstract

This report documents the compliance of testing performed on a DECsystem 5000 Model 25 server running INFORMIX-OnLine 4.10, in conformance with Revision 1.1 of the Transaction Processing Performance Council's TPC Benchmark B Standard Specification.

Two standard metrics, transactions per second (TPS) and price per TPS (K$/TPS), are reported. Throughout this report, TPS refers to the tpsB performance metric, in accordance with the TPC Benchmark B Standard.

The benchmark's methodology, results, and $/tpsB calculations were internally audited by Digital Equipment Corporation and Informix Software Inc.

Table of Contents

Preface

This report documents the compliance of the Digital TPC Benchmark B testing on a DECsystem 5000 Model 25 with the *TPC Benchmark™ B Standard Specification*[1]. The TPC Benchmark B Standard represents an effort by Digital Equipment Corporation, Informix Software Inc., and other members of the Transaction Processing Performance Council (TPC) to create an industry-wide benchmark for evaluating the performance and price/performance of transaction processing systems.

These tests were run using the INFORMIX-OnLine relational database running under the Digital ULTRIX operating system.

Document Structure

The *TPC Benchmark B Full Disclosure Report* is organized as follows:

- The main body of the document lists each item in Clause 10 of the TPC Benchmark B Standard and explains how each specification is satisfied.
- Appendix A contains the source code of the application program used to implement the TPC Benchmark B transaction and related programs and scripts.
- Appendix B contains the INFORMIX-OnLine database definitions.
- Appendix C contains the source code used to populate the database.
- Appendix D contains samples of contents of the database files used in the tests.
- Appendix E contains a description of the physical disk partitions.
- Appendix F contains the operating system parameters and options.

Additional Copies

To request additional copies of this report, write to the following address:

Administrator, TPC Benchmark Reports
Software Performance Group
Digital Equipment Corporation
151 Taylor Street (TAY1)
Littleton, MA 01460-1407
U.S.A.
FAX number: (508) 952-4197

[1] *TPC Benchmark B Standard Specification,* Transaction Processing Performance Council, March 1, 1992, Version 1.1.

TPC Benchmark B Full Disclosure

The *TPC Benchmark™ B Standard Specification* requires test sponsors to publish, and make available to the public, a full disclosure report in order for the results to be considered compliant with the standard. The required contents of the full disclosure report are specified in Clause 10.

This report is intended to satisfy the TPC Benchmark B standard's requirement for full disclosure. It documents the compliance of the benchmark tests with each item listed in Clause 10 of the *TPC Benchmark™ B Standard Specification.*

In the *TPC Benchmark™ B Standard Specification*, the main headings in Clause 10 are keyed to the other standard clauses. The headings in this report use the same sequence, so that they correspond to the titles or subjects referred to in Clause 10.

Each section in this report begins with the text of the corresponding item from Clause 10 of the *TPC Benchmark™ B Standard Specification,* printed in italic type. The plain type text that follows explains how the tests comply with the TPC Benchmark B requirement. In sections where Clause 10 requires extensive listings, the section refers to the appropriate appendix at the end of this report.

1 - General Items

1.1 Sponsor

A statement identifying the sponsor of the benchmark and any other companies who have participated.

This benchmark test was sponsored by both Digital Equipment Corporation and Informix Software, Inc.

1.2 Application Code and Definition Statements

Program listing of application code and definition language statements for files/tables. If the application environment contains software that routes or organizes the execution of transactions (e.g., a transaction processing monitor) the software must be a generally available commercial product that is fully supported as defined in Clause 9.

- Appendix A contains the C source code of the application program used to implement the TPC Benchmark B transaction and related programs and scripts.
- Appendix B contains the INFORMIX-OnLine database definitions.
- Appendix C contains the source code used to populate the database.
- Appendix D contains samples of contents of the database files used in the test.
- Appendix E contains a description of the physical disk partitions.
- Appendix F contains the operating system parameters and options.

1.3 Parameter Settings

Settings for all customer-tunable parameters and options that have been changed from the defaults found in actual products; including but not limited to:

- *Database options*
- *Recovery/commit options*
- *Consistency/locking options*
- *System parameters, application parameters, and configuration parameters*

Test sponsors may optionally provide a full list of all parameters and options.

A listing of all parameters and options is provided.

Appendixes A, B, E, and F contain the application, database configuration, partition, and operating system parameters used in the TPC Benchmark B tests.

1.4 Configuration Diagrams

Configuration diagrams of both benchmark configuration and the priced system, and a description of the differences.

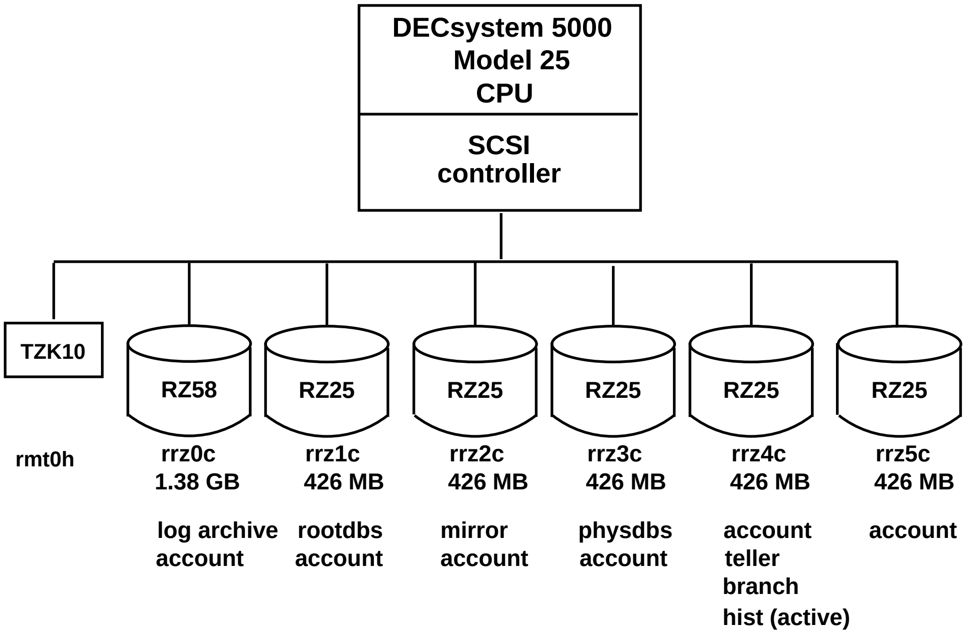
The configurations used for the benchmark and the priced system were the same.

The configuration consisted of a DECsystem 5000 Model 25 with 40 Megabytes (MB) of main memory and one embedded SCSI controller supporting one 1.38 Gigabytes (GB) RZ58 disk drive and five 426 MB RZ25 disk drives.

We enabled continuous archiving of the logical logs. The logical logs were backed up to an archive device. A 1.38 GB RZ58 disk drive was used for this purpose. This disk drive provided the necessary storage capacity so that eight hours of log data could be kept on-line. Informix transaction logging was at all times set to unbuffered mode.

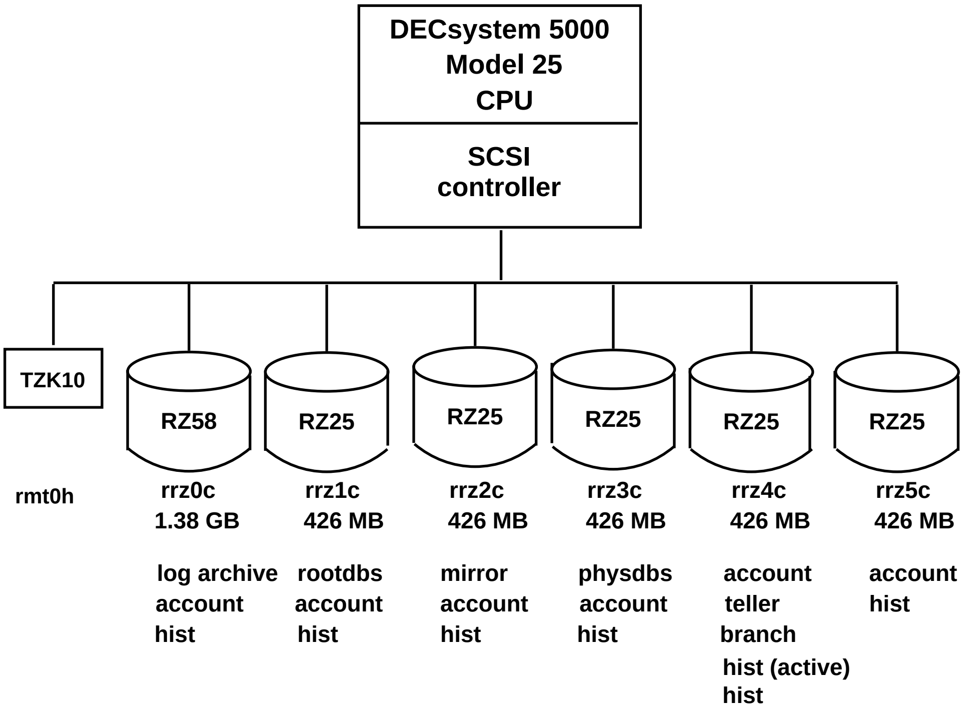
Benchmark Configuration

The diagram that follows represents the benchmark configuration.



Priced System Configuration

The diagram that follows represents the priced system configuration.



2 - Clause 2 Related Items

ACID Properties

Results of the ACIDity test (specified in Clause 2) must describe how the requirements were met. If a database different from that which is measured is used for durability tests, the sponsor must include a statement that durability works on the fully loaded and fully scaled database.

Clause 2 of the *TPC Benchmark B Standard Specification* lists specific tests to ensure the atomicity, consistency, isolation, and durability (ACID) properties of the SUT (System Under Test). The following subsections show how the tests required in Clause 2 were performed. All mechanisms needed to ensure full ACID properties were enabled during both the measurement and test periods. A fully-scaled, 24 TPS database was used for the atomicity, consistency, isolation, and instantaneous interruption and memory loss durability tests. A database sized for 3 TPS was used for durable media failure tests.

2.1 Atomicity Tests

Atomicity of Completed Transaction

Perform the standard TPC Benchmark B transaction for a randomly selected account and verify that the appropriate records have been changed in the Account, Branch, Teller, and History files/tables.

The following test was performed and verified the atomicity of completed transactions:

1. Select a random Branch record.
2. Select a random Teller record.
3. Select a random Account record.
4. For the selected Account record, count the History records and sum their delta values.
5. Using the randomly selected records, perform the following steps:
   - A. Update the Branch record.
   - B. Update the Teller record.
   - C. Update the Account record.
   - D. Insert the History record.
   - E. Commit the transaction.
   - F. Select the Branch record.
   - G. Select the Teller record.
   - H. Select the Account record.
6. For the selected Account record, count the History records and sum their delta values. Verify that the History record count and delta sum reflect the committed

transaction.

Atomicity of Aborted Transaction

Perform the standard TPC Benchmark B transaction for a randomly selected account, substituting an ABORT of the transaction account for the COMMIT of the transaction. Verify that the appropriate records have not been changed in the Account, Branch, Teller, and History files/tables.

The following test was performed, and verified the atomicity of aborted transactions:

1. Select a random Branch record.
2. Select a random Teller record.
3. Select a random Account record.
4. For the selected Account record, count the History records and sum their delta values.
5. Using the randomly selected Branch, Teller and Account records from above, do the following:
    A. Update the Branch record.

    B. Update the Teller record.

    C. Update the Account record.

    D. Insert the History record.

    E. Abort the transaction and perform a rollback recovery.

    F. Select the Branch record.

    G. Select the Teller record.

    H. Select the Account record.
6. For the selected Account record, count the History records and sum their delta values. Verify that the History record count and delta sum have not changed.

2.2 Consistency Tests

Consistency is the property of the application that requires any execution of the transaction to take the database from one consistent state to another.

The following tests were performed and verified the consistency property of transactions:

1. Consistency of Branch and Teller records before transactions
    A. Select Branch balances for each Branch record.

    B. Select the sum of Teller balances for each Branch record.

    C. Verify that the balance for each Branch record is equal to the balance of its Teller records.
2. Consistency of Branch and Teller records after transactions

   A. For the entire History file, count the History records and sum their delta values.

   B. Perform the standard TPC Benchmark B test and record the number of committed transactions.

   C. Repeat step 1.

3. Consistency of History files

   A. For the entire History file, count the History records and sum their delta values.

   B. Verify that this History record count equals the sum of History record count taken in step 2A plus the number of committed transactions.

   C. Verify that the difference between the final History delta sum and the initial History delta sum equals the difference between the final and initial Branch record balances.

2.3 Isolation Tests

Operations of concurrent transactions must yield results which are indistinguishable from the results which would be obtained by forcing each transaction to be serially executed to completion in some order.

The following tests were performed and verified the isolation property of the transactions for conventional locking used by the database system:

Isolation of Completed Transactions

1. Select the Branch balance for a Branch record (Branch B).

2. Start transaction 1.

   A. Update the Branch B record with delta(1).

   B. Stop just prior to committing transaction 1.

3. Start transaction 2.

   A. Attempt to update Branch B with delta(2).

   B. Transaction 2 hangs.

4. Resume transaction 1.

   A. Update the Teller record.

   B. Update the Account record.

   C. Insert the History record.

   D. Commit transaction 1.

5. Resume transaction 2.

   A. Update the Teller record.

   B. Update the Account record.

   C. Insert the History record.

   D. Commit transaction 2.

6. Select the Branch balance for Branch B. The balance should equal the previous balance plus delta(1) and delta(2).

Isolation of Aborted Transactions

1. Start transaction 1.

   A. Update Branch B with delta(3).

   B. Stop just prior to committing transaction 1.

2. Start transaction 2.

   A. Attempt to update Branch B with delta(4).

   B. Transaction 2 hangs.

3. Resume transaction 1.

   A. Update the Teller record.

   B. Update the Account record.

   C. Insert the History record.

   D. Abort transaction 1 and perform a rollback recovery.

4. Resume transaction 2.

   A. Update the Teller record.

   B. Update the Account record.

   C. Insert the History record.

   D. Commit transaction 2.

5. Select the Branch balance for Branch B. The balance should equal the previous balance plus delta(4).

The preceding isolation tests were repeated for Teller and Account files (that is, by generating a conflict on a Teller and then an Account record).

2.4 Durability Tests

The tested system must guarantee the ability to preserve the effects of committed transactions and ensure database consistency after recovery from any one of the failures listed below:

- *Permanent irrecoverable failure of any single durable medium containing database, ABTH (Accounts, Branch, Teller, and History) files/tables, or recovery log data.*
- *Instantaneous interruption (system crash/system hang) in processing which requires system reboot to recover.*
- *Failure of all or part of memory (loss of contents).*

The following test was performed for each of the preceding types of failures to verify the durability property of the SUT:

- For the entire History file, count the History records.
- Perform the standard TPC Benchmark B test and record the committed transactions in the success file.
- Cause one of the preceding types of failure.
- Restart the system under test for this failure as required in Clause 2.5.3.
- Verify that every record in the success file has a corresponding record in the History file.
- For the entire History file, count the History records. Verify that the number of records in the History file is greater than or equal to the original count obtained in step 1 plus the number of records in the success file. If they are different, the new History file must contain additional records and the difference must be less than or equal to the number of terminals (tellers) simulated.

In addition, the test sponsors must guarantee that, to the best of their knowledge, a fully-loaded system would pass the durability tests.

To the best of the test sponsor's knowledge, a fully-loaded and fully-scaled system would pass the durability tests.

3 - Clause 3 Related Items

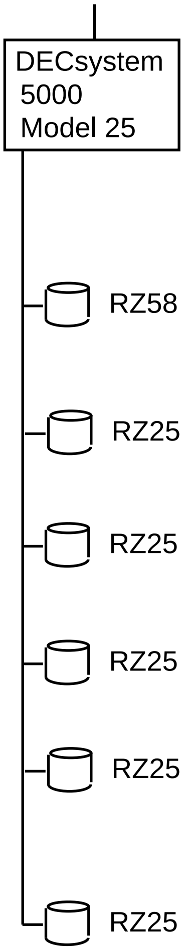
3.1 ABTH Data Storage Distribution

The distribution across storage media of ABTH (Accounts, Branch, Teller, and History) files/tables and all logs must be explicitly depicted.

The following diagram shows how the databases were distributed on disk media on the DECsystem 5000 Model 25 test system for both the benchmark and priced system configurations.

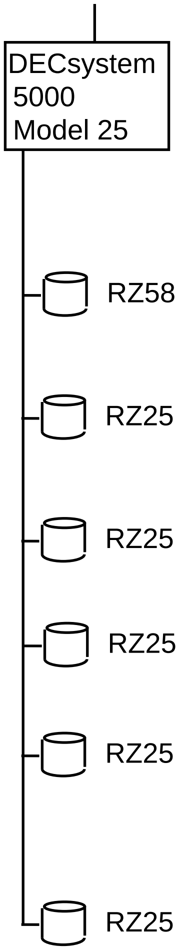
The physical log was placed on a 426 MB RZ25 disk drive and used 130 MB. The rootdbs space containing the logical logs and catalog were located on another 426 MB RZ25 disk drive and used 230 MB for the partition. The rootdbs mirror (logical log) was located on another 426 MB RZ25.

ABTH Data Storage Distribution Diagram - Benchmark Configuration



	Partition	File/ Data	Mbytes Used	Percent of Data
RZ58	rrz0h	logical log archive	837.4	100.0%
	rrz0c	account	49.8	16.6%
RZ25	rrz1c	rootdbs (logical logs)	224.6	100.0%
		account	49.8	16.6%
RZ25	rrz2c	mirror rootdbs (logical logs)	224.6	100.0%
		account	49.8	16.6%
RZ25	rrz3c	physical log	127	100.0%
		account	49.8	16.6%
RZ25	rrz4c	account	49.8	16.6%
		teller	.024	100.0%
		branch	.0024	100.0%
		history (active)	127	100.0%
RZ25	rrz5h	account	49.8	16.6%
	rrz5a	/root	9.8	
	rrz5b	swap, dump	64	
	rrz5g	/usr	134.8	

ABTH Data Storage Distribution Diagram - Priced System Configuration



	Partition	File/ Data	Mbytes Used	Percent of Data
RZ58	rrz0c	logical log archive	1060.3	100.0%
		account	49.8	16.6%
		history	179.9	18.8%
RZ25	rrz1c	rootdbs (logical logs)	24.6	100.0%
		account	49.8	16.6%
		history	131.6	13.7%
RZ25	rrz2c	mirror rootdbs (logical logs)	224.6	100.0%
		account	49.8	16.6%
		history	131.6	13.7%
RZ25	rrz3c	physical log	127	100.0%
	rrz3c	account	49.8	16.6%
		history	229.2	23.9%
RZ25	rrz4c	account	49.8	16.6%
		teller	.024	100.0%
		branch	.0024	100.0%
		history (active)	127	100.0%
		history	229.2	23.9%
RZ25	rrz5h	account	49.8	16.6%
	rrz5a	/root	9.8	
	rrz5b	swap, dump	64	
	rrz5g	/usr	134.8	
	rrz5h	history	57	5.9%

The distribution of the database is further evidenced and illustrated by the Informix tbstat utility, tbstat_d.

tbstat_d Listing

```
RSAM Version 4.10.UE1P1 -- On-Line -- Up 02:13:28 -- 12288 Kbytes

Dbspaces
address   number  flags fchunk   nchunks   flags  owner     name

80b47c    1       2     1        1         M      informix  rootdbs
80b4ac    2       1     2        1         N      informix  physdbs
80b4dc    3       1     3        1         N      informix  tbhdbs
80b50c    4       1     4        6         N      informix  acctdbs
 4 active, 10 total

Chunks
address   chk/dbs       offset   size      free   bpages  flags  pathname
8084fc    1   1         85000    115000    37678          PO-    /dev/rrz1c
809cbc    1   1         85000    115000    0              MO-    /dev/rrz2c
808594    2   2         135000   65000     9342           PO-    /dev/rrz3c
80862c    3   3         135000   65000     62630          PO-    /dev/rrz4c
8086c4    4   4         500      25500     237            PO-    /dev/rrz5h
80875c    5   4         200000   25500     497            PO-    /dev/rrz0c
8087f4    6   4         50000    25500     497            PO-    /dev/rrz2c
80888c    7   4         100000   25500     497            PO-    /dev/rrz3c
808924    8   4         100000   25500     497            PO-    /dev/rrz4c
8089bc    9   4         50000    25500     1997           PO-    /dev/rrz1c
 9 active, 40 total
```

3.1.1 History Storage and Recovery

Within the priced system, there must be sufficient on-line storage to support any expanding system files and durable history records/rows for 30 eight-hour days at the published tpsB rate (i.e., 30 x 8 x 60 x 60 = 864,000 records/rows per tpsB).

The history and log file storage calculations are shown below:

History File Storage

The following calculations were used to determine the aggregate size of the history file.

INFORMIX-OnLine Page Size	2048 bytes
Overhead per Page	28 bytes
Overhead per Row	4 bytes
History Table Row Size	50 bytes

History Rows per Page = (Page Size - Page Overhead)/(Row Size + Row Overhead)
truncated to next lowest integer value = 37 History Rows per Page

History Rows Needed = (tpsB * 3600 * 8 * 30) = 20,563,200 Rows

History Space = (Rows Needed/37) * 2048 = 1,138,200,909 Bytes

Logfile Storage

During the benchmark run, the Informix logical logs were mirrored. In addition, the inactive logfile segments were archived to disk using INFORMIX-OnLine Continuous Archiving. In all cases, unbuffered logging was used. Two disk drives were used; one for the logical logs and one for the mirror.

The Informix tbstat utility (tbstat_l) was used to record write data and logfile data production rates. In the audited reported run, the values were

Number of Writes	25,150
Pages/Write	1.2

The run had a two minute (120 seconds) ramp-up and a 26 minute measurement window. Although the number of writes occurred over the entire 28 minute period, only the steady state portion of the interval should be used for calculation because during ramp-up the log write rate would have been less. As a result, logfile space needed was as follows:

```
Total logfile storage required/8 hours=
          25,150 writes/26 minutes  * 1.2 pages/write *  2048 bytes/page ==>
          2,377,255 bytes/minute * 8 hours * 60 minutes/hour ==>
                              1,141,082,585 bytes
```

Total Logfile Space Needed:	1,141,082,585 bytes
Active Log Space Supplied	- 153,600,000 bytes

	987,482,585 bytes

Additional 8-hour log space was required. We used a 1.38 Gbyte drive to accommodate this requirement.

In addition, because INFORMIX-OnLine records a timestamp for every completed logical log archived, we used the timestamp to calculate the average time to archive one logical log during the steady state run. The average time to fill a 50000 Kbyte logical log was approximately 1560 seconds. This equates to an 8-hour logfile requirement of 945,230,769 bytes.

Because the earlier calculation showed worst case condition, we used those figures. We supplied 1,265,405,132 bytes for logical log and archive.

Informix tbstat output for the logical logs and part of the message log follow.

Clause 3 Related Items

tbstat_l listing

```
RSAM Version 4.10.UE1P1 -- On-Line -- Up 02:13:28 -- 12288 Kbytes

Physical Logging
Buffer   bufused   bufsize   numpages   numwrits     pages/io
P-1      5         128       39364      308          127.81

          phybegin physize   phypos     phyused      %used
         200292    55000     50424      6661         12.11

Logical Logging Buffer
bufused  bufsize   numrecs   numpages   numwrits recs/pages    pages/io
0        16        363757    31025      25150    11.7          1.2

address  number    flags     uniqid     begin    size          used        %used
855934   1         F------     0        100b02   25000         0           0.00
855950   2         F------     0        106caa   25000         0           0.00
85596c   3         U---C-L 12           10ce52   25000         6026        24.10
```

Message Log File Listing

```
RSAM Version 4.10.UE1P1 -- On-Line -- Up 02:13:28 -- 12288 Kbytes

Message Log File: /usr/informix/online.log
09:32:45 Checkpoint Completed
09:44:11 Logical Log 103 Complete
09:46:02 Checkpoint Completed
09:47:51 Logical Log 103 Backed Up
09:59:21 Checkpoint Completed
10:00:06 Logical Log 104 Complete
10:00:07 Checkpoint Completed
10:00:44 Logical Log 104 Backed Up
10:13:45 Checkpoint Completed
10:26:06 Logical Log 105 Complete
10:27:11 Checkpoint Completed
10:30:28 Logical Log 105 Backed Up
10:40:27 Checkpoint Completed
10:43:57 Logical Log 106 Complete
10:43:58 Checkpoint Completed
10:44:16 Logical Log 106 Backed Up
10:57:32 Checkpoint Completed
11:09:24 Logical Log 107 Complete
11:10:50 Checkpoint Completed
11:13:29 Logical Log 107 Backed Up
```

Appendix E contains a complete listing of the disk devices used to support the test.

3.2 Database Contents and Method of Population

A description of how the database was populated, along with sample contents of each ABTH file/table to meet the requirements described in Clause 3.

Appendix C contains the database population program and Appendix D contains samples of the contents of the database files used in the tests.

3.3 Type of Database

A statement of the type of database utilized, e.g., relational, Codasyl, flat file, etc.

These TPC Benchmark B tests used INFORMIX-OnLine, a relational database management system.

4 - Clause 4 Related Items

There are no Clause 4 Related Items in the checklist for TPC-B.

5 - Clause 5 Related Items

5.1 Method of Verification of Random Number Generator

The method of verification of the random number generator should be described.

Branch, Teller, and Account IDs were generated by the random number generation routines, random() and srandom() in the bench.h code. Random()/srandom() use a non-linear additive feedback random number generator, employing a default table size of 31 long integers to return successive random numbers in the range from 0 to (2**31)-1. These routines produce a more random sequence than earlier subroutines such as rand(). Random() and srandom() are well known random number generation routines. Randomness of the generated values are further verified by observing the 85/15 distribution rule, which showed that approximately 85% of the transactions submitted to a Branch had the Account belong to that Branch.

5.2 Horizontal Partitioning Disclosure

Vendors must clearly disclose if horizontal partitioning is used. Specifically, vendors must:

- *Describe textually the extent of transparency of the implementation*
- *Which tables/files were accessed using partitioning*
- *How partitioned tables/files were accessed*

The intent of this clause is that details of non-transparent partitioning be disclosed in a manner understandable to non-programmer individuals (through use of flow charts, pseudo code, etc.).

Horizontal partitioning, i.e., the partitioning of a table among different devices, was used. The account relation records were evenly distributed over six disk drives. A single history table was used.

5.3 Transaction Distribution

The sponsor must disclose percentage of remote and home transactions, percentage of remote and foreign transactions, if applicable, and the actual distribution of accounts across the nodes, if applicable.

The measured percentage of remote transactions for the test was 15%. This was done by querying the history records using the SQL script "RANDVERIFY.SQL" which verified that approximately 15% of the accounts were from a different branch as required by the TPC Benchmark B Standard.

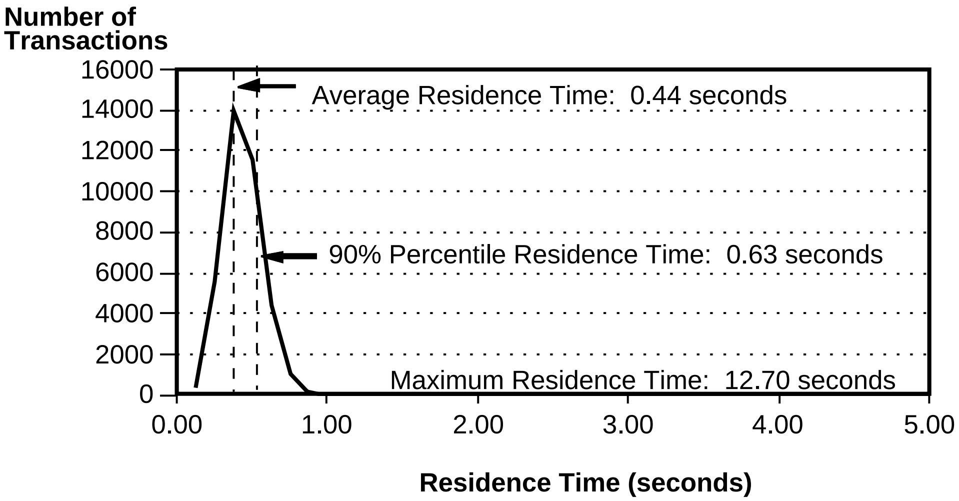
6 - Clause 6 Related Items

Report all the data specified in Clause 6.6, including maximum and average residence time, as well as performance curves for number of transactions vs. residence time (see Clause 6.6.1) and throughput vs. level of concurrency for three data points (see Clause 6.6.5). Also, the sponsor must include the percentage of home and remote transactions, the number and percentage of in-process transactions, and the percentage of remote and foreign transactions, if applicable.

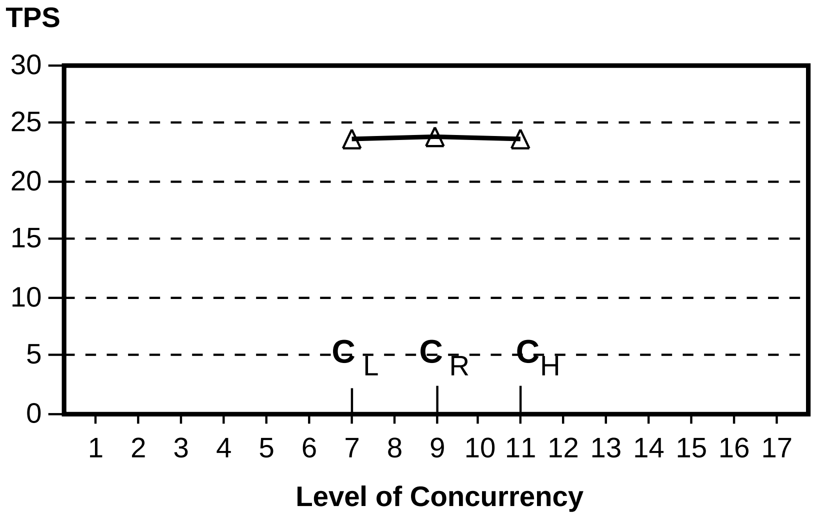
The graphs and tables in this section show the residence time performance results as well as the percentage of home and remote transactions, and in-process transactions. There are no foreign transactions.

Please note that for all performance runs the database was scaled for 25 TPS.

Residence Time Frequency Distribution for All Transactions



Throughput Versus Level of Concurrency



Concurrency Legend

Measured Points	Level of Concurrency	TPS	Average Residence Time (seconds)
C_L	6.8	23.7	0.29
C_R	9.0	23.8	0.38
C_H	10.9	23.7	0.46

Profile of Executed Transactions

Description	Result
Remote Transactions (see Clause 6.6.2)	15.00%
Home Transactions	85.00%
Transactions started and not completed during measurement interval (see Clause 6.6.3)	0.02%
Number of transaction started but not completed	9
Total number of transactions	37,199
Average residence time for all transactions	0.38 seconds
Maximum residence time for all transactions	12.70 seconds
Percent of all transactions qualified within 2 second response time constraint	99.95
Maximum qualified throughput	23.8 tpsB

7 - Clause 7 Related Items

7.1 Determining Steady State

The method used to determine that the SUT had reached a steady state prior to commencing the measurement interval should be described.

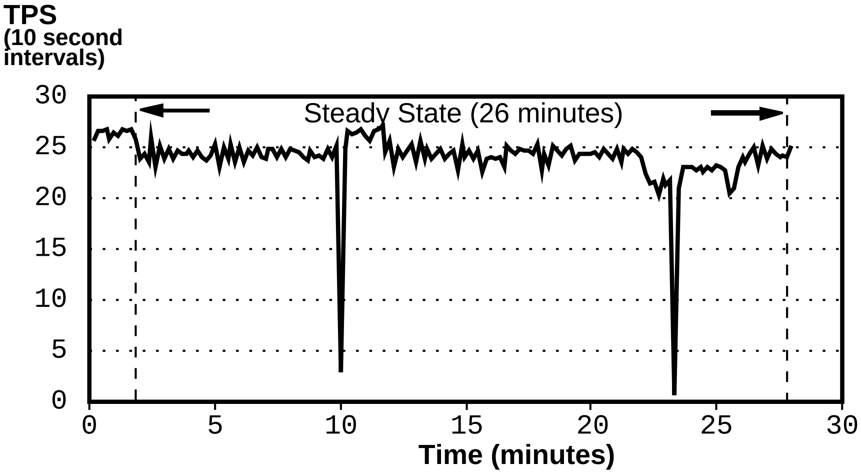
Confirmation that the SUT has reached steady state prior to the beginning of the data collection measurement interval is based on a visual inspection of the plot of TPS versus time.

The design of the benchmark driver program was such that all processes wait to be signaled to commence ramp-up work. During ramp-up, the processes begin executing identical TPC-B transactions as they do during the steady state run.

During the ramp-up, which lasted for 120 seconds, all processes began executing the identical TPC-B transaction that they do during the timed steady state run. At the end of the ramp-up period, each process independently kept track of the numbers and characteristics of its committed transactions that started during the steady state interval. The audited benchmark steady state period lasted for 26 minutes. When the run was completed, the processes individually and independently reported their accumulated transactions and residence time results. The driver then calculated the required numbers to report.

To confirm that steady state was reached, the history table was examined. The graph titled "TPS Versus Time" indicates the number of transactions completed in each 10 second interval. The steady state portion is labeled on the graph. Note the pronounced dips (checkpoints) in transaction rate that occurred two times during the steady state for the run.

TPS Versus Time



7.2 Work Performed During Steady State

A description of how the work normally performed during a sustained test (for example, checkpointing, writing redo/undo log records, etc., as required by Clause 7.2), actually occurred during the measurement interval.

When INFORMIX-OnLine receives a SQL statement from the application, it determines how to best access the data. Using an index (B-tree), INFORMIX-OnLine determines the page number from the database that the record is located on, and searches for the page in shared memory.

If the page is not in shared memory, INFORMIX-OnLine chooses a LRU Buffer in shared memory and reads the page from the database into the buffer. Typically this will take two disk reads. The first read acquires the bottom level of the B-tree index, the second disk read actually acquires the data.

When a transaction starts, a BEGIN WORK is written to the logical log buffers. When the application issues a SQL UPDATE statement (Account, Teller, and Branch) to modify a record, the copy of the record, if it is already in shared memory, is locked and updated. A transaction record is written to the logical log buffer.

At the same time, if the page in shared memory has not previously been written to, a copy of the before image of the page is written to the physical log buffer in shared memory. In addition, the before and after images of the record are written to the logical log buffer in shared memory. So, the physical log buffer contains a copy of the page that a record is on, as it looked prior to making any modification.

When the application issues a SQL COMMIT WORK statement, the logical log buffer is flushed from shared memory to the logical log on disk in a single I/O. The database pages remain in shared memory and are not written to the database at that time. Any locks that were placed by the transaction are released. This means that when an application commits a transaction to the database, the logical log buffer is written to a corresponding logical log on disk with a single I/O, and successful completion code is returned to the application.

Periodically INFORMIX-OnLine will automatically write all modified pages in shared memory to their appropriate locations in the database during a checkpoint. A checkpoint is preceded by a write of the physical log buffer to the physical log on disk. Checkpoints occur periodically during the run. With INFORMIX-OnLine there are several ways of controlling when a checkpoint occurs. For our benchmark, checkpointing occurs every time INFORMIX-OnLine starts the last logical log. We configured INFORMIX-OnLine with three logical logs. Thus, every time two logs were filled and the third started, a checkpoint occurred. In our benchmark run of 26 minutes, 2 checkpoints occurred.

When the checkpoint occurs, one or more background processes called page cleaners "wake up" and write all the modified pages from shared memory to the database on disk. A checkpoint record is written to the logical log buffer. A checkpoint message is written to the message log.

The page reading and writing activity to the individual chunk partitions in the database are reflected in the Informix utility tbstat_D and tbstat_p.

tbstat_D

```
RSAM Version 4.10.UE1P1 -- On-Line -- Up 02:13:28 -- 12288 Kbytes

Dbspaces
address   number flags   fchunk  nchunks  flags    owner     name
80b47c    1      2       1       1        M        informix  rootdbs
80b4ac    2      1       2       1        N        informix  physdbs
80b4dc    3      1       3       1        N        informix  tbhdbs
80b50c    4      1       4       6        N        informix  acctdbs
  4 active, 10 total

Chunks
address   chk/dbs  offset     page Rd    page Wr    pathname
8084fc    1    1   85000      25003      31029      /dev/rrz1c
809cbc    1    1   85000      0          31029      /dev/rrz2c
808594    2    2   135000     0          39360      /dev/rrz3c
80862c    3    3   135000     1514       1096       /dev/rrz4c
8086c4    4    4   500        13188      6308       /dev/rrz5h
80875c    5    4   200000     13068      6297       /dev/rrz0c
8087f4    6    4   50000      13275      6324       /dev/rrz2c
80888c    7    4   100000     13057      6249       /dev/rrz3c
808924    8    4   100000     12724      6095       /dev/rrz4c
8089bc    9    4   50000      11999      5745       /dev/rrz1c
   9 active, 40 total
```

tbstat_p

```
RSAM Version 4.10.UE1P1 -- On-Line -- Up 02:13:28 -- 12288 Kbytes

Profile
dskreads pagreads bufreads  %cached dskwrits pagwrits bufwrits %cached
80619    103830   1176960   93.15   88449    139529   163913   46.04

isamtot  open     start     read    write    rewrite  delete   commit  rollbk
1491997  55       121276    121487  40434    121217   0        40572   0

ovtbls   ovlock   ovuser    ovbuff  usercpu  syscpu   numckpts flushes
0        0        0         0       919.98   471.60   2        4

bufwaits lokwaits lockreqs  deadlks dltouts  lchwaits ckpwaits compress
3857     5035     945369    0       0        19086    18       1
```

7.3 Determining Reproducibility

A description of the method used to determine the reproducibility of the measurement results.

Experiments were repeated at least 3 times at the maximum targeted TPS level to ensure reproducibility. The results are shown in the following table. The variation in TPS was less than 1%.

DECsystem 5000 Model 25 TPC-B Benchmark Runs

Run #	Processes	CPUs	tpsB	Percent < 2 sec.	Transactions	db Size	Duration
1	9	1	23.8	99.95	37,199	25 tps	26.0 mins
2	9	1	23.6	99.95	36,918	25 tps	26.0 mins
3	9	1	23.8	99.95	37,193	25 tps	26.0 mins

7.4 Duration of Measurement Period

A statement of the duration of the measurement period for the reported tpsB (it should be at least 15 minutes and no longer than 1 hour).

Each experiment used a measurement period of 26 minutes and began approximately

2 minutes after all servers had begun executing transactions.

8 - Clause 8 Related Items

8.1 Description of the Driver

If the driver is commercially available, then its inputs should be specified. Otherwise, a description of the driver should be supplied.

The driver used was an "internal driver" (i.e., the driver software resides on the system under test, not on a remote driver machine) that controls transaction processing and performance data collection for the TPC Benchmark B runs. The driver was comprised of two parts: a control csh script and a set of identical ESQL/C transaction programs that submitted the TPC Benchmark B transactions for execution.

The control script performs the following operations:

1. forks and execs the desired number of transaction programs, passing ramp-up and measurement interval parameters as command line arguments.
2. waits for a short period of time (30 seconds) to ensure that each driver has started up and opened the test database.
3. sends a SIGUSR1 signal to each transaction process to synchronize the start of transaction processing.
4. waits until all transaction processes have completed the benchmark run.
5. invokes a program called sumrun to sum the performance statistics collected by the transaction processes involved in the benchmark run.

   After each transaction program completes a benchmark run, the transaction program stores residence time counts, incomplete transaction counts, and other performance statistics in a database table named "results". The sumrun program sums all "results" records for a run and inserts aggregate run values into a table named "runs".

Each transaction program performs the following operations:

1. examines its command line arguments to determine the ramp-up and measurement intervals to use.
2. waits until it receives a SIGUSR1 signal before initiating transaction processing.
3. continuously submits TPC-B transactions, with 0 sleep time. The transaction program collects response time statistics in internal program data structures, but does not begin collecting them until the ramp-up period has completed.
4. inserts its collected performance statistics into a "results" table record once the measurement interval has completed. It is the contents of these "results" records that are summed by the sumrun program.

"Success files" were implemented through the tpc.ec application program by writing synchronously using fsync() and flushing the confirmation of transactions to standard output. This was captured into a file nohup.out running under the Korn shell.

9 - Clause 9 Related Items

9.1 Hardware and Software Components

A detailed list of hardware and software used in the priced system. Each item must have vendor part number, description, and release/revision level, and either general availability status or committed delivery date. If package pricing is used, contents of the package must be disclosed.

9.1.1 Priced System Configuration Tables

The following table shows the hardware and software components in the priced DECsystem 5000 Model 25 system:

Component	Product	Quantity
Processor	DECsystem 5000 Model 25	1
Memory		40 Megabytes
Tape drive	TZK10	1
Disk controller	SCSI	1
Disks	RZ58	1
	RZ25	5
Operating system	ULTRIX 4.2A	
Database	INFORMIX-OnLine V4.10 INFORMIX ESQL/C	

9.1.2 Availability Status

All hardware and software components used in the tested and priced systems are available now.

9.1.3 Package Pricing

Package Description	Model #
DECsystem 5000 Model 25	PM319-RX
Server System Kernel	
120 Volt or 240 Volt	
24 MB, 4 Mbit DRAMS, 25 Mhz CPU	
425 MB RZ25 Disk Drive	
SCSI Controller	
Thickwire Ethernet	
Includes SCSI Cable, Terminal Cable	
Hardware Documentation	
Includes Software Licenses: ULTRIX 1-4 user, UWS server support	

Package Description	Model #
Two RZ25 426MB Disk/BA42 120V	SZ12G-GA

Package Description	Model #
1.38 RZ58 Disk Drive, TZK10 Quarter-Inch Cartridge (QIC) Tape Device	SZ12J-EA

9.2 Total Price of System Configuration

The total price of the entire configuration is required including: hardware, software and maintenance charges. Separate component pricing is recommended. The basis of all discounts used shall be disclosed.

This section lists the separate components in the priced system and their associated purchase and maintenance costs. All items are currently available. All prices were taken from the Digital Standard Pricing System (DSPS) on March 24, 1992. A description of the packages used in the pricing is contained in Section 9.1.2.

Informix prices were taken from Informix price list, titled "America's Price List, Advance Products, Release 4.0 or Greater, Class D", dated August 1, 1991.

9.2.1 Hardware Pricing

The Digital TPC Benchmark B DECsystem 5000 Model 25 test used packaged hardware systems whenever possible to simplify configurations to the fewest number of line items. Disks were connected using SCSI controller. The system used a TZK10 tape drive to load the software and back up the database.

The purchase price of all systems includes one year of hardware warranty service. Post-warranty hardware service is configured for an additional four years.

The following levels of post-warranty hardware service are used in the system pricing:

- DECsystem Support 9x5 (DS9) and 2-4 hours response time.
- Basic Monthly Charge (BMC) warranty level is the same as the DS9 to which the hardware is directly attached.
- Basic System Support (BSS) with a warranty upgrade to DS9.

9.2.2 Software Pricing

The priced system uses the following software products:

- ULTRIX V4.2A operating system
- INFORMIX-OnLine relational database management system
- INFORMIX-ESQL/C

The ULTRIX license purchase includes one year of warranty service. Post-warranty service is configured for an additional four years. The software warranty and service level are the same as the service level for the hardware system on which the software operates. The level of post-warranty service is Software Support Service (SSS).

9.2.3 Price Discounts

Digital's five (5) years warranty pricing is as follows:

- the unit price carries one (1) year warranty.
- the price of year 2-5 warranty adder is calculated according to this formula:
  - (warranty/month)*12*(1+1+1.07+$(1.07)^2$)=(1.053725*48*(warranty/month))

The pre-payment maintenance (warranty) discount is calculated at 25% of the year 2-5 warranty price.

Informix's five-year prepaid maintenance option consists of five years of maintenance for four times the price of standard maintenance.

9.2.4 System Pricing Summary

DECsystem 5000 Model 25 TPC-B = 23.8 tpsB

US LIST DESCRIPTION	MODEL #	UNIT PRICE 1 YR WARR	QTY	TOTAL PRICE	SERVICE LEVEL	MAIN. $/MO.	# MO	2-5 YRS MAIN. PRICE	PRICE+SRVC 5 YR COST
Digital Price (24 March 1992):									
Hardware									
DECsystem 5000 Model 25 (24 MB)	PM319-RX	$12,659.00	1	$12,659.00	BSS	$0.00	48	$0.00	$12,659.00
Warranty Upgrade to DS9	FM-DECUP-12	$132.00	1	$132.00	DS9	$173.00	48	$8,304.00	$8,436.00
16 MB Memory Units	MS01-CA	$3,200.00	1	$3,200.00	DS9/BMC	$0.00	48	$0.00	$3,200.00
Two 426 MB RZ25 Disks	SZ12G-GA	$5,600.00	2	$11,200.00	DS9/BMC	$50.00	48	$4,800.00	$16,000.00
One 1.38 GB RZ58, TLZ04 Tape Drive	SZ12J-EA	$9,284.00	1	$9,284.00	DS9/BMC	$82.00	48	$3,936.00	$13,220.00
Software									
ULTRIX	ULTRIX V4.2	$0.00	1	$0.00	NA	$0.00	48	$0.00	$0.00
UX-32 Media & Documentation	QA-VYVAA-H5	$1,315.00	1	$1,315.00	NA	$0.00	48	$0.00	$1,315.00
				$37,790.00				$17,040.00	$54,830.00
Years 2-5 Warranty Adder = 5.3725%								$915.47	$915.47
Digital Sub Total				$37,790.00				$17,955.47	$55,745.47
Prepayment Maintenance Discount = (25%)								($4,488.87)	($4,488.87)
Digital Total				$37,790.00				$13,466.60	$51,256.60
Informix Class "D" License (1 AUGUST 1991):									
Database System						$/YR	YR		
INFORMIX-OnLine (16U)	FULL DEV./RUN T	$6,700.00	1	$6,700.00	SSS	$1,210.00	4	$4,840.00	$11,540.00
INFORMIX-ESQL/C (16U)	FULL DEV./RUN T	$1,340.00	1	$1,340.00	SSS	$240.00	4	$960.00	$2,300.00
Informix Total				$8,040.00				$5,800.00	$13,840.00
Configuration Totals				$45,830.00				$19,266.60	$65,096.60
								tpsB	23.8
								$/tpsB	$2,735

9.3 Performance and Price/Performance

A statement of the measured tpsB, and the calculated price/tpsB.

The following table shows measured tpsB and price/tpsB results for the tested system:

CPU Model	Software	TPS (tpsB)	Price per TPS ($/tpsB)
DECsystem 5000 Model 25	ULTRIX 4.2A and INFORMIX-OnLine 4.10	23.8	$2,735

10 - Clause 10 Related Items

None.

Appendix A

Application Code

This appendix contains the source code of the application programs that implement the TPC Benchmark B transaction.

A.1   tpc.ec Source Code

```
#include <stdio.h>
#include <sys/signal.h>
#include <sys/types.h>
#include <sys/timeb.h>
#include <math.h>
$include sqlca ;
#include "bench.h"

$long acct_bal, cntr, seconds, intvl, startsec, tot_response ;
$int branch_num, teller_num, acct_num, delta, acct_branch, run, procnum ;
$int notdone, tmslot[BUCKETS+1] ;
int rampup, runtime, timing, thru, measure, bucketval, transactions, verbose ;
int longest_tran;

settimer() {  timing = ~timing ; }

setmeasure()
{
        intvl = (measure) ? rampup : runtime ;
        thru++ ;
        measure = ~measure ;
         startsec = time(0) ;
}

main(argc,argv)
        int argc ;
        char **argv ;
{
        int i, *rnum, do_trans() ;

         runtime = rampup = intvl = 0 ;
        transactions = -1 ;
        procnum = atoi(argv[1]) ;
        i = 1 ;

         while (++i < argc) {
                if (strcmp(argv[i], "-s") == 0)
                runtime += atoi(argv[++i]) ;
                else if (strcmp(argv[i], "-m") == 0)
                runtime += (60 * atoi(argv[++i])) ;
                else if (strcmp(argv[i], "-h") == 0)
                runtime += (3600 * atoi(argv[++i])) ;
                else if (strcmp(argv[i], "-t") == 0)
                transactions = atoi(argv[++i]) ;
                else if (strcmp(argv[i], "-r") == 0)
                rampup = atoi(argv[++i]) ;
                else if (strcmp(argv[i], "-v") == 0)
                verbose = 1 ;
                else {
                        fprintf(stderr,"usage: tp1 <proc #> [-t <n>] [-r <n>]
[-h <n> -m <n> -s <n>]\n") ;
                exit(1) ;
                }
 }

        RandSeed(getpid()) ;

        if (runtime == 0)
                runtime = (transactions == -1) ? 300 : 30000 ;

        printf("process %d: procnum=%03d runtime=%d seconds / %d transac
                tions\n", getpid(),procnum,runtime,transactions) ;

        cntr = tot_response = run = measure = timing = thru = intvl = notdone = 0 ;
        for (i=0; i < 50; i++)
                tmslot[i] = 0 ;

        $ database tpc ;
           SqlErr("attach to database") ;

        $ select max(number) into $run from results ;
           SqlErr("select from results") ;
        if (run < 0)
                run = 0 ;
         ++run ;

         do_prepare() ;

         bucketval = RPTINTVL * 1000 / BUCKETS ;
        intvl = rampup ;
        signal(SIGUSR1,settimer) ;
        sigpause(0) ;

        printf("%d starting\n",procnum) ;  do_trans() ;  testend() ;

}

do_prepare()
{
        $char s[512];

            sprintf(s,"%s %s %s%d%s %s commit work",
            "update account set balance = balance + ? where current of sel_acct;",
            "update teller set balance = balance + ? where number = ?;",
            "insert into history",procnum % HISTORY,
            " values(?,?,?,?,CURRENT YEAR TO SECOND,’the rest is history’);",
            "update branch set balance = balance + ? where number = ?;") ;
         $ prepare tpc_trans from $s;
          SqlErr("prepare updall");

         $ prepare bwork from "begin work" ;
          SqlErr("prepare begin work") ;

         $ declare sel_acct cursor for
                select balance into $acct_bal from account
                        where number = $acct_num
                        for update of balance ;
          SqlErr("declare cursor") ;

         $ set isolation to cursor stability ;
          SqlErr("set isolation") ;

         $ set lock mode to wait;
          SqlErr("set lock mode");
}

do_trans()
{
        long timediff ;
        char s[100] ;
        struct timeb clk_beg,clk_end ;

                startsec = time(0) ;
        if (rampup == 0)
                setmeasure() ;
        else
                thru++ ;

         while (timing && (cntr != transactions)) {

            /*
             * select a random branch, a random teller at that branch, and
             * 85% of the time a random account at that branch, and 15% of
             * the time a random acccount at a different branch.
             */

        teller_num = RandVal() % T_RECS ;
        branch_num = teller_num / T_PERB ;
        acct_num = RandVal() % A_PERB ;

        if ((RandVal() % 100) < 85)
                acct_branch = branch_num ;
        else {
                do              /* endless loop when TPS_SIZE=1 */
                        acct_branch = RandVal() % B_RECS ;
                 while (acct_branch == branch_num) ;
        }

        acct_num = acct_branch * A_PERB + acct_num ;
        delta = RandVal() % 1999999 - 999999 ;
        if (measure)
                 notdone++ ;
```

```
ftime(&clk_beg) ;

$ execute bwork ;

$ open sel_acct ;
   SqlErr("open cursor") ;

$ fetch sel_acct ;
   SqlErr("fetch cursor") ;

if (sqlca.sqlcode == 0) {
 $ execute tpc_trans using
                    $delta,
                    $delta, $teller_num,
                    $acct_num, $teller_num, $branch_num, $delta,
                    $delta, $branch_num ;
}

ftime(&clk_end) ;

if (sqlca.sqlcode != 0) {
         sprintf(s,"in transaction %d acc#: %d branch#: %d teller#: %d",
                                        cntr, acct_num, branch_num, teller_num) ;
 SqlFatal(s) ;
 /*
 $ rollback work ;
 */
}

timediff = clk_end.time - startsec ;
if (timediff > intvl) {
                    if (thru == 2)
                                        settimer() ;
                     else
                                        setmeasure() ;
}

if (measure) {
        timediff = (clk_end.time - clk_beg.time) * 1000
                                        + clk_end.millitm - clk_beg.millitm ;
 if(timediff > longest_tran)
        longest_tran = timediff;
 tot_response += timediff ;
 timediff /= bucketval ; /* 0-.124, .125-0.249, etc. seconds */
 if (timediff > BUCKETS)
         timediff = BUCKETS ;
 tmslot[timediff]++ ;
 cntr++ ;
 if(verbose)
 {
        printf("procnum %3d: tran %d completed!\n",procnum,cntr);
        fflush(stdout);
        fsync(1);
 }
```

```
             notdone-- ;
            }

        }

        seconds = (transactions > 0) ? (time(0)-startsec) : runtime ;

}

testend()
{
        int hrs, min, sec ;

            hrs = seconds / 3600 ;
            min = (seconds - hrs * 3600) / 60 ;
            sec = seconds - hrs * 3600 - min * 60 ;
            printf("procnum %3d completed %6d transactions in %4d:%02d:%02d, longest=%d msec.\n",
                        procnum, cntr, hrs, min, sec, longest_tran) ;

            $ insert into results values(
                          $run, $procnum, $seconds, $cntr, $notdone, $tot_response,
                          $tmslot[0],$tmslot[1],$tmslot[2],$tmslot[3],$tmslot[4],
                          $tmslot[5],$tmslot[6],$tmslot[7],$tmslot[8],$tmslot[9],
                          $tmslot[10],$tmslot[11],$tmslot[12],$tmslot[13],$tmslot[14],
                          $tmslot[15],$tmslot[16],$tmslot[17],$tmslot[18],$tmslot[19],
                          $tmslot[20],$tmslot[21],$tmslot[22],$tmslot[23],$tmslot[24],
                          $tmslot[25],$tmslot[26],$tmslot[27],$tmslot[28],$tmslot[29],
                          $tmslot[30],$tmslot[31],$tmslot[32],$tmslot[33],$tmslot[34],
                          $tmslot[35],$tmslot[36],$tmslot[37],$tmslot[38],$tmslot[39],
                          $tmslot[40]) ;
                          SqlErr("insert into results") ;

            $ close database ;
              SqlErr("close database") ;
 }
```

A.2   createdb.ec Source Code

```
#include <stdio.h>
#include "bench.h"
 $include sqlca ;

/*
* FILE: createdb.ec (for OnLine)
*
* Creates the database and related tables, except result-consolidation
* tables. It is possible to place the tables on different drives by
* adding location options to the CREATE TABLE statements.
*
* You can also decide to place logging on the database by adding it
* to the CREATE DATABASE statement. However, the loading programs
* provided assume no transaction logging, so you should turn on logging
* afterward via archiving and changing the database logging mode.
*
* The configuration here accommodates scaling to 100 TPS.
*
*/

main()
{

                $ create database tpc in TBHDBS ;
                  SqlErr("create database") ;

                $ grant dba to public ;
                   SqlErr("grant dba") ;

                printf("Database created, permission granted\n") ;

                $ create table branch      (
                                number numeric(2,0),
                                balance numeric(10,0),
                                fillstr char(92)
                )
                lock mode row
                ;
                 SqlErr("create branch") ;
                printf("Branch created\n") ;

                $ create table teller
                (
                                number numeric(4,0),
                                balance numeric(10,0),
                                branch numeric(2,0),
                                fillstr char(89)
                )
                extent size 200
                next size 100
                lock mode row
```

```
                    ;
                      SqlErr("create teller") ;
                    printf("Teller created\n") ;

                    $ create table account      (
                    number          numeric(8,0),
                    balance         numeric(10,0),
                    branch          numeric(2,0),
                    fillstr          char(87)
                    )
                    in acctdbs
                    extent size 5000
                    next size 1000
                    ;
                      SqlErr("create account") ;
                    printf("Account created\n") ;

                    $ close database ;
                      SqlErr("close database") ;

                     exit(0) ;
}
```

A.3   createhist.ec Source Code

```
#include <stdio.h>
#include "bench.h"
 $include sqlca ;

/*
* FILE: createhist.ec (for OnLine)
*
* Creates the history tables. Number of tables is HISTORY in "bench.h".
*
* The configuration here accommodates scaling to 100 TPS.
*
*/

main()
{
        $char dstr[200] ;
        $int cnt, i ;

         $ database tpc ;
           SqlErr("connect to database") ;

        $ select count(*) into $cnt from systables
                where tabname matches "hist" ;
          SqlErr("test for history tables") ;

        $ select count(*) into $cnt from systables
                where tabname matches "hist*" ;
          SqlErr("test for history tables") ;

        if (cnt) {
                printf("Dropping History tables...\n") ;
                for (i=0; i < cnt; i++) {
                        sprintf(dstr,"drop table history%d",i) ;
                        $ prepare drop_tab from $dstr ;
                          SqlErr("prepare drop") ;
                        $ execute drop_tab ;
                          SqlErr(dstr) ;
                }
         }

        for (i=0; i < HISTORY; i++) {

                sprintf(dstr, "%s%d (%s,%s,%s,%s,%s,%s) %s %s %s",
                        "create table history", i,
                                "account integer",
                                "teller integer",
                                "branch integer",
                                "delta char(11)",
                                "tstamp datetime year to second",
                                "fillstr char(19)",
                                "extent size 1000",
                                "next size 1000",
                                                  "lock mode row"
                                     ) ;
                         $ prepare make_tab from $dstr ;
                           SqlErr("prepare create") ;
                         $ execute make_tab ;
                           SqlErr("execute history") ;
                         printf("History%d table created\n",i) ;

 }

 $ close database ;
   SqlErr("close database") ;

 exit(0) ;
}
```

A.4   createruns.ec Source Code

```
#include <stdio.h>
#include "bench.h"
$include sqlca ;

/*
 * FILE:  createruns.ec
 *
 * Creates the results tables for cumulative reporting
 *
 */

main()
{
    $int cnt ;

                     $ database tpc ;
                        SqlErr("open database") ;

                    $ select count(*) into $cnt from systables
                                        where tabname = "runs" ;
                        SqlErr("test for runs table") ;

                    if (cnt) {
                                        $ drop table runs ;
                                           SqlErr("drop table runs") ;
                    }

                    $ select count(*) into $cnt from systables
                                        where tabname = "results" ;
                       SqlErr("test for results table") ;

                    if (cnt) {
                                        $ drop table results ;
                                           SqlErr("drop table results") ;
 }

    $ create table runs
                    (
                     num           serial,
                     numprocs      integer,
                     test_intvl    integer,
                     total_xact    integer,
                     total_inc     integer,
                     resp_time     integer,
                     cpus          integer,
                     test_size     integer,
                     tslot01       integer,
                     tslot02       integer,
                     tslot03       integer,
                     tslot04       integer,
                     tslot05       integer,
                     tslot06       integer,
                     tslot07      integer,
                     tslot08      integer,
                     tslot09      integer,
                     tslot10      integer,
                     tslot11      integer,
                     tslot12      integer,
                     tslot13      integer,
                     tslot14      integer,
                     tslot15      integer,
                     tslot16      integer,
                     tslot17      integer,
                     tslot18      integer,
                     tslot19      integer,
                     tslot20      integer,
                     tslot21      integer,
                     tslot22      integer,
                     tslot23      integer,
                     tslot24      integer,
                     tslot25      integer,
                     tslot26      integer,
                     tslot27      integer,
                     tslot28      integer,
                     tslot29      integer,
                     tslot30      integer,
                     tslot31      integer,
                     tslot32      integer,
                     tslot33      integer,
                     tslot34      integer,
                     tslot35      integer,
                     tslot36      integer,
                     tslot37      integer,
                     tslot38      integer,
                     tslot39      integer,
                     tslot40      integer,
                     tslot41      integer
                    ) ;
        SqlErr("create runs") ;
printf("Runs table created\n") ;

        $ create table results    (
                     number       integer,
                     procnum      integer,
                     seconds      integer,
                     xactcnt      integer,
                     notdone      integer,
                     response     integer,
                     tslot01      integer,
                     tslot02      integer,
                     tslot03      integer,
                     tslot04      integer,
                     tslot05         integer,
                     tslot06         integer,
                     tslot07         integer,
                     tslot08         integer,
                     tslot09         integer,
                     tslot10         integer,
                     tslot11         integer,
                     tslot12         integer,
                     tslot13         integer,
                     tslot14         integer,
                     tslot15         integer,
                     tslot16         integer,
                     tslot17         integer,
                     tslot18         integer,
                     tslot19         integer,
                     tslot20         integer,
                     tslot21         integer,
                     tslot22         integer,
                     tslot23         integer,
                     tslot24         integer,
                     tslot25         integer,
                     tslot26         integer,
                     tslot27         integer,
                     tslot28         integer,
                     tslot29         integer,
                     tslot30         integer,
                     tslot31         integer,
                     tslot32         integer,
                     tslot33         integer,
                     tslot34         integer,
                     tslot35         integer,
                     tslot36         integer,
                     tslot37         integer,
                     tslot38         integer,
                     tslot39         integer,
                     tslot40         integer,
                     tslot41         integer
                    ) ;
    SqlErr("create results") ;
 printf("Results table created\n") ;

 $ close database ;
    SqlErr("close database") ;

 exit(0) ;
}
```

A.5 createidx.ec Source Code

```
#include <stdio.h>
#include "bench.h"
$include sqlca ;

/*
 * FILE: createidx.ec
  *
  * Creates the indices for the main database tables. This is a separate
  * process in case loads without indices are desired.
  *
  */

main()
 {

                    $ database tpc ;
                       SqlErr("open database") ;

                    $ create unique index ibranch on branch(number) ;
                       SqlErr("create branch index") ;  printf("Branch index created\n") ;

                    $ create unique index iteller on teller(number) ;
                       SqlErr("create teller index") ;  printf("Teller index created\n") ;

                    $ create unique index iaccount on account(number) ;
                       SqlErr("create account index") ;
                     printf("Account index created\n") ;

                    $ close database ;
                       SqlErr("close database") ;

                     exit(0) ;
}
```

A.6 config.scr Source Code

```
echo Going into Quiescent mode
tbmode -uy
echo Creating physdbs...
tbspaces -c -d physdbs -p /dev/rrz3c -o 270000 -s 130000
echo Creating tbhdbs...
tbspaces -c -d tbhdbs -p /dev/rrz4c -o 270000 -s 130000
echo Creating acctdbs...
tbspaces -c -d acctdbs -p /dev/rrz5h -o 1000 -s 51000
echo Adding chunk to acctdbs...
tbspaces -a acctdbs -p /dev/rrz0c -o 400000 -s 51000
echo Adding chunk to acctdbs...
tbspaces -a acctdbs -p /dev/rrz2c -o 100000 -s 51000
echo Adding chunk to acctdbs...
tbspaces -a acctdbs -p /dev/rrz3c -o 200000 -s 51000
echo Adding chunk to acctdbs...
tbspaces -a acctdbs -p /dev/rrz4c -o 200000 -s 51000
echo Adding chunk to acctdbs...
tbspaces -a acctdbs -p /dev/rrz1c -o 100000 -s 51000
echo Moving Physical Log
tbparams -p -s 110000 -d physdbs -y
echo Going back On-Line
tbmode -m
 echo Configuration done
```

A.7   bench.h code

```
*
 PURPOSE: to set up the sizing of the TPC database
*
* the scale factors for TPC per 1 TPS are:
 *              1 Branch, 10 Tellers, 100000 Accounts
 *
* Modify the TPS_SIZE to the desired rating.
* DO NOT modify any but the first 4 lines.  *
*/

#define TPS_SIZE          25
#define HISTORY           1
#define RandVal           random
#define RandSeed          srandom
#define BUCKETS           40
#define RPTINTVL          5

#define T_PERB            10
#define A_PERB             100000

#define B_RECS             TPS_SIZE
#define T_RECS             (T_PERB * B_RECS)
#define A_RECS             (A_PERB * B_RECS)

#define IsqlCode          sqlca.sqlcode
#define IsamCode          sqlca.sqlerrd[1]
#define SqlErr(x)         if (IsqlCode) Sqlmsg(x)
#define SqlErrNF(x)       if (IsqlCode && IsqlCode != SQLNOTFOUND) Sqlmsg(x)
```

Appendix B

Database Definitions

```
#******************************************************************************************
#
#                           INFORMIX SOFTWARE, INC.
#
# Title:                    tbconfig.std
# Sccsid:                   @(#)tbconfig.std 7.2 11/20/90 11:06:55
#Description:               INFORMIX-OnLine Configuration Parameters
#
#******************************************************************************************

# Root Dbspace Configuration

ROOTNAME                    rootdbs     # Root dbspace name
ROOTPATH                    /dev/rrz1c  # Path for device containing root dbspace
ROOTOFFSET                  170000      # Offset of root dbspace into device (Kbytes)
ROOTSIZE                    230000      # Size of root dbspace (Kbytes)

# Disk Mirroring Configuration Parameters

MIRROR                      1           # Mirroring flag (Yes = 1, No = 0)
MIRRORPATH                  /dev/rrz2c  # Path for device containing mirrored root
MIRROROFFSET                170000      # Offset into mirrored device (Kbytes)

# Physical Log Configuration

PHYSDBS                     physdbs     # Location (dbspace) of physical log
PHYSFILE                    110000      # Physical log file size (Kbytes)

# Logical Log Configuration

LOGFILES                    3           # Number of logical log files
LOGSIZE                     50000       # Logical log size (Kbytes)

# Message Files

MSGPATH                     /usr/informix/online.log     # System message log file path
CONSOLE                     /usr/informix/console.log    # System console message path

# System Archive Tape Device

TAPEDEV                     /dev/null   # Tape device path
TAPEBLK                     16          #  Tape block size (Kbytes)
TAPESIZE                    90000       #  Maximum  amount  of  data  to  put  on  tape
                                        #   (Kbytes)
# Log Archive Tape Device

LTAPEDEV          /dev/rrz0h   # Log tape device path
LTAPEBLK          16           # Log tape block size (Bytes)
LTAPESIZE         1000472      # Max amount of data to put on log tape (Kbytes)

# System Configuration

SERVERNUM         0            # Unique id corresponding to an OnLine instance
SERVERNAME        dectpc       #
DEADLOCK_TIMEOUT  30           # max time to wait of lock in distributed env.
RESIDENT          0            # Forced residency flag (Yes = 1, No = 0)

# Shared Memory Parameters

USERS             50           #  Maximum  number  of  concurrent  users  (proc
                               #   esses)
LOCKS             5000         # Maximum number of locks
BUFFERS           5000         # Maximum number of shared buffers
TBLSPACES         500          # Maximum number of open tblspaces
CHUNKS            40           # Maximum number of chunks
DBSPACES          10           # Maximum number of dbspaces
PHYSBUFF          256          # Physical log buffer size (Kbytes)
LOGBUFF           32           # Logical log buffer size (Kbytes)
LOGSMAX           5            # Maximum number of logical log files
CLEANERS          8            # Number of buffer cleaner processes
SHMBASE           0x800000     # Shared memory base address
CKPTINTVL         780          # Check point interval (in sec)

# System Page Size

BUFFSIZE          2048         # Page size (do not change!)
```

Appendix C

Code to Populate Database

This appendix contains the program used to populate the database used in the TPC Benchmark B tests.

C.1 Database Population Program

The following programs were used to populate the database.

C.1.1 load_db.ec Source Code

```
#include <stdio.h>
#include <math.h>
#include <sys/types.h>
#include <sys/wait.h>
#include "bench.h"
$include sqlca ;
/*
 * FILE: load_db.ec
 *
 * PURPOSE: load the Branch and Teller tables, and kick off the Account
 *           table load procedures. The Account table is loaded by
 *           dividing the key range into equal parts (according to the
 *           number of load processes), and the "load_act" program is
 *           forked off for each process. The program then waits for
 *           them to finish and reports the total load time.
 *
 * NOTE: The type "pid_t" may be system-dependent. Under Ultrix it’s
 * equivalent to an "int".
 *
 */

FILE *flog,*fopen() ;
int logfile ;

main(argc,argv)
          int argc ;
          char *argv[] ;
{
          int i, load_procs, skip, freespace ;
          char begnum[15], endnum[15], log_fname[40], rpt_str[80] ;
          long load_accts, startacct, acct_hunk, beg_time, end_time, totsecs ;
          pid_t pid ;
          union wait wait_status ;

          $int branch, teller, branch_idx ;
          $char filler[100] ;
```

```
                RandSeed(getpid()) ;
                load_procs = 1 ;
                i = logfile = skip = freespace = branch_idx = 0 ;

                while (++i < argc) {
                        if (strcmp(argv[i], "-p") == 0)
                                load_procs = atoi(argv[++i]) ;
                        else if (strcmp(argv[i], "-s") == 0)
                                skip = atoi(argv[++i]) ;
                        else if (strcmp(argv[i], "-f") == 0)
                                freespace = atoi(argv[++i]) ;
                        else if (strcmp(argv[i], "-l") == 0) {
                                strcpy(log_fname,argv[++i]) ;
                                logfile = 1 ;
                        }
                        else {
                                printf("usage: load_db -p <#> -s <#> -
f <#> -l <file>\n") ;
                        exit(0) ;
                        }
                 }

                if (load_procs && ((A_PERB % load_procs) != 0)) {
                        printf("Cannot split up load of accounts evenly. Try
again.\n") ;
                        exit(0) ;
                }

                load_accts = (B_RECS - skip) * A_PERB ;
                        if (load_procs) acct_hunk = load_accts /
load_procs ;
                for (i=0; i < 10; i++)
                        bycopy("1234567890",&filler[i*10],10) ;
                if (logfile) {  if ((flog=fopen(log_fname,"w")) == NULL) {
                        perror("on opening log file") ;
                        logfile = 0 ;
                }
        }

        $ database tpc ;
          SqlErr("database open") ;

        if (freespace) {
                $ select count(*) into $branch_idx from sysindexes
                        where idxname = "ibranch" ;
                   SqlErr("load_db -- select branch index") ;
                if (branch_idx) {
                        $ drop index ibranch ;
                           SqlErr("load_db -- delete branch index") ;
                }
        }

        for (branch=skip; branch < B_RECS; branch++) {
                $ insert into branch values($branch, 0, $filler) ;
                  SqlErr("load_db -- branch insert") ;
                for (i=0; i < freespace; i++) {
                        $ insert into branch values(0, -1, $filler) ;
                          SqlErr("load_db -- branch free insert") ;
                }
}
if (freespace) {
                $ delete from branch where balance < 0 ;
                  SqlErr("load_db -- delete branch records") ;
 }
if (branch_idx) {
                $ create unique index ibranch on branch(number) ;
                  SqlErr("load_db -- create branch index") ;
}
 print_log("branch table loaded") ;
for (teller=T_PERB*skip; teller < T_RECS; teller++) {
                branch = teller / T_PERB ;
                $ insert into teller values($teller, 0, $branch, $filler) ;
                  SqlErr("load_db -- insert into teller") ;
 }
 print_log("teller table loaded") ;
$ update statistics for table branch ;
  SqlErr("load_db -- update stats on branch") ;
$ update statistics for table teller ;
  SqlErr("load_db -- update stats on teller") ;
$ close database ;
  SqlErr("load_db -- close database") ;
sqlexit() ;
if (load_procs) {
                beg_time = time(0) ;
                startacct = skip * A_PERB ;
                for (i=0; i < load_procs; i++) {
                        sprintf(begnum,"%d",startacct) ;
                        startacct += acct_hunk ;
                        sprintf(endnum,"%d",startacct-1) ;
                        pid = fork() ;
                         if (pid == -1) {
                                perror("on fork of loadact process") ;
                                exit(1) ;
                }
                if (pid == 0)
                if (logfile)
                 execl("load_act","load_act",begnum,endnum,"1",log_fname,0) ;
                else
```

```
                 execl("load_act","load_act",begnum,endnum,"0"," ",0) ;
                }
                while (i--) {
                        pid = wait(&wait_status) ;
                        if (pid == -1) {
                                perror("on return from loadact") ;
                                exit(1) ;
                        }
                        end_time = time(0) ;
                        totsecs = end_time - beg_time ;
                        sprintf(rpt_str,"\nprocess %d completed in ",pid) ;
                        report_time(rpt_str,totsecs) ;
                        if (i > 0)
                                sprintf(rpt_str,"%s; %d procs still
working",rpt_str,i) ;
                         print_log(rpt_str) ;
                }
                sprintf(rpt_str,"\nAll processes finished at
%s",ctime(&end_time)) ;
                print_log(rpt_str) ;
                sprintf(rpt_str,"loaded %d account records in ",load_accts) ;
                report_time(rpt_str,totsecs) ;
                sprintf(rpt_str,"%s = %d rows/sec\n",rpt_str,load_accts/totsecs)
;
                print_log(rpt_str) ;
                $ database tpc ;
                  SqlErr("Open Database") ;
                $ update statistics for table account ;
                  SqlErr("Update Statistics on account") ;
                $ close database ;
                  SqlErr("Close Database") ;
         }
         if (logfile)   fclose(flog) ;
        exit(0) ;
}
report_time(s,secs)
        char s[] ;
        long secs ;
{
        int hrs, mins, slen ;
        hrs = secs / 3600 ;
        secs = secs % 3600 ;
        mins = secs / 60 ;
        secs = secs % 60 ;
        slen = strlen(s) ;
        sprintf(&s[slen],"%2d:%02d:%02d",hrs,mins,secs) ;
```

```
}
print_log(s)
            char *s ;
{
            if (logfile) {
                        fprintf(flog,"%s\n",s) ;
                        fflush(flog) ;
 }
else {
                        printf("%s\n",s) ;
                        fflush(stdout) ;
            }
}
```

C.1.2   load_act.ec Source code

```
#include <stdio.h>
#include <sys/time.h>
#include <sys/signal.h>
#include <math.h>
#include "bench.h"
$include sqlca ;

/*
* FILE: load_act.ec
*
* PURPOSE: multi-process load of the Account table; the key range of
*       the accounts to load is passed.
 *
*/

FILE *flog,*fopen() ;
int logfile, kill_ld ;

main(argc,argv)
        int argc ;
        char *argv[] ;
{
        int i, setstop() ;
        long begnum, endnum, firstlog, loginterval, logcnt, to_go, done ;
        long begsecs, endsecs ;
        float rate, last_rate, slow_down ;
        char log_fname[40], rpt_str[100], time_str[30] ;
        $int branch ;
        $long account ;
        $char filler[100] ;

                rstol(argv[1],&begnum) ;
                rstol(argv[2],&endnum) ;
                logfile = atoi(argv[3]) ;
                strcpy(log_fname,argv[4]) ;

                for (i=0; i < 10; i++)
                        bycopy("1234567890",&filler[i*10],10) ;
                if (logfile) {
                        if ((flog=fopen(log_fname,"a")) == NULL) {
                                perror("on opening log file") ;
                                logfile = 0 ;
                        }
                 }

/*
* try to keep processes from reporting at the same time, report
* 7-10 times, report fairly soon for initial rate
*/
```

```
last_rate = 0.0 ;
slow_down = 1.0 ;
to_go = endnum - begnum + 1 ;
loginterval = to_go / (7 + getpid() % 4) ;
 rate = 20.0 + (getpid() % 6) * 10.0 ;
firstlog = begnum + loginterval * rate / 100 ;
sprintf(rpt_str,"proc  %d:  loading  records  %d  -  %d,  reporting  every  %d",
        getpid(),begnum,endnum,loginterval) ;
print_log(rpt_str) ;

kill_ld = 0 ;
signal(SIGUSR1,setstop) ;                         /* in case you need to kill_load */

$ database tpc ;
  SqlErr("database open") ;

$ declare ins_acct cursor for
        insert into account values( $account, $branch, 0, $filler) ;
 SqlErr("declare cursor") ;

$ open ins_acct ;
  SqlErr("first open") ;

begsecs = time(0) ;
logcnt = 0 ;

for (account=begnum; account <= endnum && !kill_ld; account++) {

        branch = account / A_PERB ;
                $ put ins_acct ;
                   SqlErr("put into account") ;

        if ((account % 500) == 0) {
                $ close ins_acct ;
                 SqlErr("periodic close of cursor") ;
                $ open ins_acct ;
                 SqlErr("periodic open of cursor") ;
        }

        if (++logcnt == loginterval || account == firstlog) {
                endsecs = time(0) ;
                done = account - begnum + 1 ;
                to_go = endnum - account ;
                strcpy(time_str,ctime(&endsecs)) ;
                time_str[24] = 0 ;
                sprintf(rpt_str,"\nproc %d: completed %d rows on %s",
                        getpid(),done,time_str) ;
                print_log(rpt_str) ;
                rate = 1.0 * (endsecs - begsecs) ;
                rate = done / rate ;
                 if (last_rate > 0.0)
                        slow_down = rate / last_rate ;
                last_rate = rate ;
                endsecs = endsecs + (to_go / rate / slow_down) ;
```

```
                        strcpy(time_str,ctime(&endsecs)) ;
                        time_str[24] = 0 ;
                        sprintf(rpt_str,"proc %d: rate is now %.1f rows/sec, e.t.c. is %s",
                                getpid(),rate,time_str) ;
                        print_log(rpt_str) ;
                        if (logcnt == loginterval)
                                logcnt = 0 ;
                }
        }

        $ close ins_acct ;  SqlErr("final close of cursor") ;

        $ close database ;  SqlErr("close database on account") ;

        if (logfile)
                fclose(flog) ;

        exit(0) ;
}

setstop()
{
        kill_ld = 1 ;
}

        report_time(s,secs)
        char s[] ;
        long secs ;
{
        int hrs, mins, slen ;

                hrs = secs / 3600 ;
                secs = secs % 3600 ;
                mins = secs / 60 ;
                secs = secs % 60 ;
                slen = strlen(s) ;  sprintf(&s[slen],"%2d:%02d:%02d",hrs,mins,secs) ;
}

print_log(s)
        char *s ;
        {
                if (logfile) {
                        fprintf(flog,"%s\n",s) ;
                        fflush(flog) ;
                }
                else {
                        printf("%s\n",s) ;
                        fflush(stdout) ;
                }
}
```

Appendix D

Database Contents Samples

This appendix contains the database contents samples for the TPC Benchmark B run on the DECsystem 5000 Model 25.

D.1 Branch Table

Following is a sample of the Branch table contents:

```
number 3
balance -51355309
fillstr 123456789012345678901234567890123456789012345678901234567890
12345678901234567890123456789012

number 4
balance -20818451
fillstr 123456789012345678901234567890123456789012345678901234567890
12345678901234567890123456789012

number 5
balance 89491712
fillstr 123456789012345678901234567890123456789012345678901234567890
12345678901234567890123456789012
```

D.2 Teller Table

Following is a sample of the Teller table contents:

```
number 8
balance -10125134
branch 0
fillstr 123456789012345678901234567890123456789012345678901234567890
12345678901234567890123456789

number 9
balance -3469726
branch 0
fillstr 123456789012345678901234567890123456789012345678901234567890
12345678901234567890123456789

number 10
balance 15177678
branch 1
fillstr 123456789012345678901234567890123456789012345678901234567890
12345678901234567890123456789
```

D.3 History Table

Following is a sample of the History table contents:

```
account 976082
teller 39
branch 3
delta 942596
tstamp 1992-03-28 03:49:22
 fillstr the rest is history

account 2478956
teller 244
branch 24
delta 690436
tstamp 1992-03-28 03:49:22
fillstr the rest is history

account 1435695
teller 142
branch 14
delta -329829
tstamp 1992-03-28 03:49:22
fillstr the rest is history
```

D.4 Account Table

Following is a sample of the Account table contents:

```
number 8
balance 0
branch 0
fillstr 123456789012345678901234567890123456789012345678901234567890
12345678912345678901234567

number 9
balance 0
branch 0
fillstr 123456789012345678901234567890123456789012345678901234567890
123456789012345678901234567

number 10
balance 0
branch 0
fillstr 123456789012345678901234567890123456789012345678901234567890
123456789012345678901234567
```

Appendix E

Device Configurations

This appendix contains a description of the physical disk configurations tested for the DECsystem 5000 Model 25 configuration.

```
/dev/rrz0a
No partition table found in superblock...
using default table from device driver.
Current partition table:
partition   bottom      top         size        overlap
 a          0           32767       32768       c
 b          32768       163839      131072      c
 c          0           2698060     2698061     a,b,d,e,f,g,h
 d          163841      1008640     844800      c,g,h
 e          1008641     1853440     844800      c,h
 f          1853441     2698060     844620      c,h
 g          163841      983040      819200      c,d
 h          983042      2698060     1715019     c,d,e,f

/dev/rrz1a
No partition table found in superblock...
using default table from device driver.
Current partition table:
partition   bottom      top         size        overlap
 a          0           32767       32768       c,h
 b          32768       163839      131072      c
 c          0           832526      832527      a,b,d,e,f,g,h
 d          163840      386735      222896      c,g
 e          386736      609631      222896      c,g
 f          609632      832526      222895      c,g
 g          163840      832526       668687     c,d,e,f
 h          0           0           0           a,c

/dev/rrz2a
No partition table found in superblock...
using default table from device driver.
Current partition table:
partition   bottom      top         size        overlap
 a          0           32767       32768       c,h
 b          32768       163839      131072      c
 c          0           832526      832527      a,b,d,e,f,g,h
 d          163840      386735      222896      c,g
 e          386736      609631      222896      c,g
 f          609632      832526      222895      c,g
 g          163840      832526      668687      c,d,e,f
 h          0           0           0           a,c
```

```
/dev/rrz3a
No partition table found in superblock...
using default table from device driver.
Current partition table:
partition    bottom       top          size         overlap
 a           0            32767        32768        c,h
 b           32768        163839       131072       c
 c           0            832526       832527       a,b,d,e,f,g,h
 d           163840       386735       222896       c,g
 e           386736       609631       222896       c,g
 f           609632       832526       222895       c,g
 g           163840       832526       668687       c,d,e,f
 h           0            0            0            a,c

/dev/rrz4a
No partition table found in superblock...
using default table from device driver.
Current partition table:
partition    bottom       top          size         overlap
 a           0            32767        32768        c,h
 b           32768        163839       131072       c
 c           0            832526       832527       a,b,d,e,f,g,h
 d           163840       386735       222896       c,g
 e           386736       609631       222896       c,g
 f           609632       832526       222895       c,g
 g           163840       832526       668687       c,d,e,f
 h           0            0            0            a,c


/dev/rrz5a
Current partition table:
partition    bottom       top          size         overlap
 a           0            32767        32768        c
 b           32768        163839       131072       c
 c           0            832526       832527       a,b,d,e,f,g,h
 d           163840       386735       222896       c,g
 e           386736       609631       222896       c,g
 f           609632       832526       222895       c,g,h
 g           163840       613839       450000       c,d,e,f
 h           613840       832526       218687       c,f
```

Appendix F

System Parameter Settings

This appendix contains the operating system parameters and database options in the TPC Benchmark B test system.

F.1 System Parameters

ULTRIX version 4.2 system parameters were configured as shown below. In all instances default values were used except for

- MAXUSERS was set to 1024
- MAXUPRC was set to 1024
- SMMAX was set to 1024
- SMSEG was set to 256

Additionally, two semaphore constant values were changed in the ULTRIX IPC Semaphore Facility sem.h (/usr/sys/h/sem.h). The value SEMMNI, the number of semaphore identifiers, was set to 80, and the SEMMNS, the number of semaphores in the system, was set to 300. A copy of sem.h appears in this appendix.

The following operating system parameters were used for the test system.

```
ident                     "HAMMRD"
machine                   mips
cpu                       "DSPERSONAL_DECSTATION"
maxusers     1024
processors   1
maxuprc                   1024
smmax                     1024
smseg                     256
physmem                   40
timezone                  5 dst 1

options                   QUOTA
options                   INET
options                   NFS
options                   RPC
options                   LAT
options                   DLI
options                   UFS
options                   NETMAN

makeoptions               ENDIAN="-EL"

config                    vmunix       root on rz5a swap on rz5b dumps on rz5b
```

```
adapter         ibus0 at nexus?
adapter         ibus1 at nexus?
adapter         ibus3 at nexus?
controller      asc0            at ibus?        vector ascintr
disk            rz0             at asc0         drive 0
disk            rz1             at asc0         drive 1
disk            rz2             at asc0         drive 2
disk            rz3             at asc0         drive 3
disk            rz4             at asc0         drive 4
disk            rz5             at asc0         drive 5
disk            rz6             at asc0         drive 6
disk            rz7             at asc0         drive 7
tape            tz0             at asc0         drive 0
tape            tz1             at asc0         drive 1
tape            tz2             at asc0         drive 2
tape            tz3             at asc0         drive 3
tape            tz4             at asc0         drive 4
tape            tz5             at asc0         drive 5
tape            tz6             at asc0         drive 6
tape            tz7             at asc0         drive 7
controller      asc1            at ibus?        vector ascintr
disk            rz8             at asc1         drive 0
disk            rz9             at asc1         drive 1
disk            rz10            at asc1         drive 2
disk            rz11            at asc1         drive 3
disk            rz12            at asc1         drive 4
disk            rz13            at asc1         drive 5
disk            rz14            at asc1         drive 6
disk            rz15            at asc1         drive 7
tape            tz8             at asc1         drive 0
tape            tz9             at asc1         drive 1
tape            tz10            at asc1         drive 2
tape            tz11            at asc1         drive 3
tape            tz12            at asc1         drive 4
tape            tz13            at asc1         drive 5
tape            tz14            at asc1         drive 6
tape            tz15            at asc1         drive 7
controller      asc2            at ibus?        vector ascintr
disk            rz16            at asc2         drive 0
disk            rz17            at asc2         drive 1
disk            rz18            at asc2         drive 2
disk            rz19            at asc2         drive 3
disk            rz20            at asc2         drive 4
disk            rz21            at asc2         drive 5
disk            rz22            at asc2         drive 6
disk            rz23            at asc2         drive 7
tape            tz16            at asc2         drive 0
tape            tz17            at asc2         drive 1
tape            tz18            at asc2         drive 2
tape            tz19            at asc2         drive 3
tape            tz20            at asc2         drive 4
tape            tz21            at asc2         drive 5
tape            tz22            at asc2         drive 6
tape            tz23            at asc2         drive 7
device          fd0             at ibus?        vector fdintr
device          dti0            at ibus?        vector dtiintr
device          ln0             at ibus?        vector lnintr
device          scc0            at ibus?        vector sccintr

scs_sysid                       1

pseudo-device                   pty
pseudo-device                   nfs
pseudo-device                   rpc
pseudo-device                   loop
pseudo-device                   lta
pseudo-device                   lat
pseudo-device                   dli
pseudo-device                   ether
pseudo-device                   ufs
pseudo-device                   netman
pseudo-device                   inet

pseudo-device                   tc
```

F.2   IPC Semaphore Facility

```
/* @(#)sem.h                4.1 (ULTRIX)             7/2/90 */
/************************************************************************
 *                                                                      *
 *              Copyright (c) 1986, 1988 by                             *
 *              Digital Equipment Corporation, Maynard, MA              *
 *              All rights reserved.                                    *
 *                                                                      *
 * This software is furnished under a license and may be used and       *
 * copied only in accordance with the terms of such license and         *
 * with the inclusion of the above copyright notice. This               *
 * software or any other copies thereof may not be provided or          *
 * otherwise made available to any other person. No title to and       *
 * ownership of the software is hereby transferred.                     *
 *                                                                      *
 * This software is derived from software received from the             *
 * University of California, Berkeley, and from Bell                    *
 * Laboratories. Use, duplication, or disclosure is subject to          *
 * restrictions under license agreements with University of             *
 * California and with AT&T.                                            *
 *                                                                      *
 * The information in this software is subject to change without       *
 * notice and should not be construed as a commitment by Digital        *
 * Equipment Corporation.                                               *
 *                                                                      *
 * Digital assumes no responsibility for the use or reliability         *
 * of its software on equipment which is not supplied by Digital.       *
 *                                                                      *
 ************************************************************************/
/*  *  * Modification history:
 *
 * 19 Mar 90 -- burns
 *              Added ifdef kernel around SMP lock imbedded in
 *              a user visable data structure (msqid_ds).
 *
 * 13 Dec 89 -- scott
 *              xpg compliance changes
 *
 * 16 Aug 88 -- miche
 *              Add support for SMP
 *
 * 02 Apr 86 -- depp
 *              Moved sizing constants from /sys/h/param.h to here.
 *
 * 01 Mar 85 -- depp  *     New file derived from System V IPC
 *
 */
/*
 **		IPC Semaphore Facility.
*/
#ifndef KERNEL
#include <sys/smp_lock.h>
extern int semctl();
extern int semget();
extern int semop();
#endif /* KERNEL */
#if !defined(_POSIX_SOURCE)
/*
**		Implementation Constants.
*/
#define	PSEMN	(PZERO + 3)		/* sleep priority waiting for greater value */
#define	PSEMZ	(PZERO + 2)		/* sleep priority waiting for zero */
/*
**		Permission Definitions.
*/
#define	SEM_A	0200		/* alter permission */
#define	SEM_R	0400		/* read permission */
#endif /* !defined(_POSIX_SOURCE) */
/*
**		Semaphore Operation Flags.
*/
#define	SEM_UNDO010000	/* set up adjust on exit entry */
/*
**		Semctl Command Definitions.
*/
#define	GETNCNT	3		/* get semncnt */
#define	GETPID	4		/* get sempid */
#define	GETVAL	5		/* get semval */
#define	GETALL	6		/* get all semval's */
#define	GETZCNT	7		/* get semzcnt */
#define	SETVAL	8		/* set semval */
#define	SETALL	9		/* set all semval's */
/*
**		Structure Definitions.
*/
/*
**		There is one semaphore id data structure for each set of semaphores
**				in the system. The ipc_perm structure must be first and
```

```
**                              the lock must be last.
*/

struct semid_ds {
                struct ipc_perm                 sem_perm;   /* operation permission struct */
                struct sem    *sem_base;        /* ptr to first semaphore in set */
                unsigned short                  sem_nsems;/* # of semaphores in set */
                time_t                          sem_otime;  /* last semop time */
                time_t                          sem_ctime;  /* last change time */
#ifdef KERNEL
                struct __lock_t sem_lk;         /* SMP lock for the semaphore queue */
#endif /* KERNEL */ };

/*
**              There is one semaphore structure for each semaphore in the system.
*/

struct sem {
                unsigned short semval;          /* semaphore text map address */
                pid_t sempid;                   /* pid of last operation */
                unsigned short semncnt;         /* # awaiting semval > cval */
                unsigned short semzcnt;         /* # awaiting semval = 0 */
                unsigned short semnwakup;/* wake up those waiting on semncnt */
};

#if !defined(_POSIX_SOURCE)

/*
**              There is one undo structure per process in the system.
*/

struct sem_undo {
                struct sem_undo                 *un_np;         /* ptr to next active undo structure */
                short                           un_cnt;         /* # of active entries */
                struct undo {
                                short           un_aoe;         /* adjust on exit values */
                                short           un_num;         /* semaphore # */
                                int             un_id;          /* semid */
                }               un_ent[1];      /* undo entries (one minimum) */
};

/*
** semaphore information structure
*/
struct          seminfo         {
                int             semmap,         /* # of entries in semaphore map */
                                semmni,         /* # of semaphore identifiers */
                                semmns,         /* # of semaphores in system */
                                semmnu,         /* # of undo structures in system */
                                semmsl,         /* max # of semaphores per id */
                                semopm,         /* max # of operations per semop call */
                                semume,         /* max # of undo entries per process */
                                semusz,         /* size in bytes of undo structure */
```

```
                        semvmx,     /* semaphore maximum value */
                        semaem;     /* adjust on exit max value */
};

/*
**          User semaphore template for semop system calls.
*/

struct sembuf {
            unsigned short sem_num;             /* semaphore # */
            short       sem_op;                 /* semaphore operation */
            short       sem_flg;                /* operation flags */
};

/*
 * Sizing constants
 */

#define SEMMAP 10
#define SEMMNI 80
#define SEMMNS 300
#define SEMMNU 30
#define SEMMSL 25
#define SEMOPM 10
#define SEMUME 10
#define SEMVMX 32767
#define SEMAEM 16384

#endif /* !defined(_POSIX_SOURCE) */
```