Technical Report ITL-99-2
June 1999

US Army Corps of Engineers
Waterways Experiment Station

Performance Testing of CEFMS

by William A. Ward, Jr., University of South Alabama

Prepared for Headquarters, U.S. Army Corps of Engineers

Technical Report ITL-99-2
June 1999

Performance Testing of CEFMS

by William A. Ward, Jr.

Faculty Court West 20
School of Computer and Information Sciences
University of South Alabama
Mobile, AL 36688

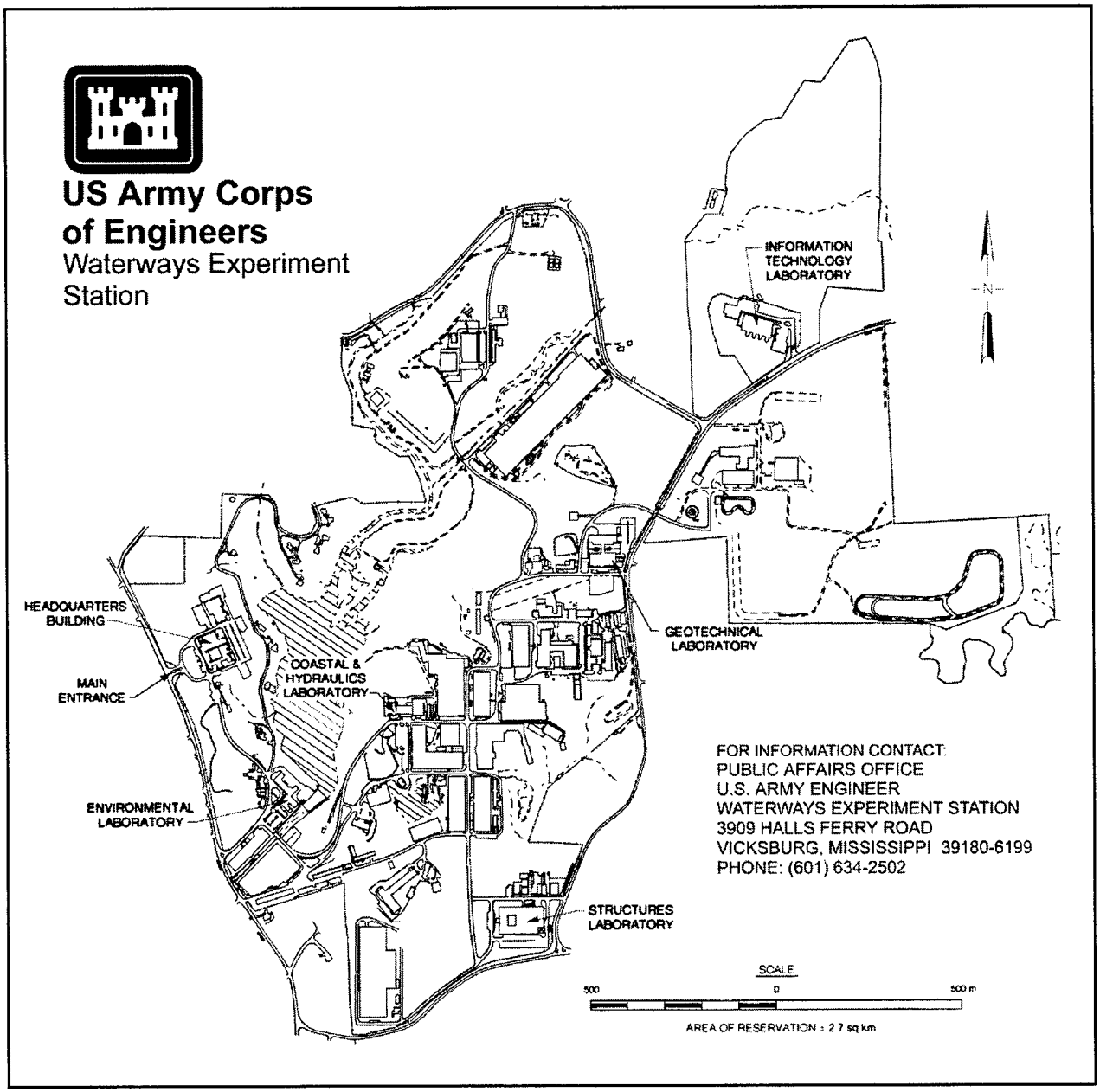
Final report

Approved for public release; distribution is unlimited

Prepared for U.S. Army Corps of Engineers
Washington, DC 20314-1000

Under Contract No. DACA39-93-K-0016

Monitored by U.S. Army Engineer Waterways Experiment Station
3909 Halls Ferry Road, Vicksburg, MS 39180-6199



Waterways Experiment Station Cataloging-in-Publication Data

Ward, William A.

Performance testing of CEFMS / by William A. Ward, Jr. ; prepared for U.S. Army Corps of Engineers ; monitored by U.S. Army Engineer Waterways Experiment Station.

93 p. : ill. ; 28 cm. — (Technical report ; ITL-99-2)

Includes bibliographical references.

1. CEFMS (Computer program) — Testing. 2. United States — Army — Corps of Engineers — Finance. 3. United States — Army — Corps of Engineers — Accounting. I. United States. Army. Corps of Engineers. II. U.S. Army Engineer Waterways Experiment Station. III. Information Technology Laboratory (U.S. Army Engineer Waterways Experiment Station) IV. Title. V. Series: Technical report (U.S. Army Engineer Waterways Experiment Station) ; ITL-99-2.

TA7 W34 no.ITL-99-2

Contents

List of Figures

List of Tables

Preface

The production of this report was sponsored by the Corps of Engineers Automation Plan (CEAP) Program Office and funded through the U.S. Army Engineer Waterways Experiment Station (WES) under Contract No. DACA39-93-K-0016 from 1 January 1995 to 30 June 1995. The contract was monitored by Dr. Windell F. Ingram, Chief, Computer Science Division (CSD), Information Technology Laboratory (ITL), WES. Dr. N. Radhakrishnan was Director, ITL.

Dr. William A. Ward, Jr., University of South Alabama, prepared this report. Mr. Howard S. Gary, Computing and Communications Center, ITL; Ms. Sherry L. Klein, Directorate of Resource Management, WES; and Ms. Elaine H. Johnson and Mr. Wallace D. Pratt, CSD, provided technical information and advice essential to the study. Mr. Gary also provided the section on SUT Disks and Database Layout in Appendix C. Those actively participating in the evluation effort were: Messrs. David Gamble, Jerry A. Graham, Lew Harkins, Clyde Hill, and Otis N. James, CSD, and Dr. Ward.

At the time of publication of this report, COL Robin R. Cababa, EN, was Acting Director of WES.

1 Introduction

The CEFMS Project

The Corps of Engineers Financial Management System (CEFMS) is designed to handle all types of financial data processing within the Corps, including processing of time sheets, travel requests, purchase requests, and in-house labor requests. It is intended to be an interactive system giving managers and principal investigators immediate access to current account information formerly available only in voluminous, and perhaps already dated, paper reports. Furthermore, it incorporates state-of-the-art security mechanisms; in addition to the standard password protection provided by UNIX,[1] magnetically coded signature cards issued only to selected individuals must be used with CEFMS to actually authorize expenditures. Judged on the basis of features provided, CEFMS is in many respects a truly modern FMS.

In spite of, or perhaps because of, its modern features, CEFMS places heavy demands on the host hardware platforms (currently a variety of Sun SPARCservers) and sometimes responds sluggishly to interactive requests. The slowness of CEFMS experienced at several operational sites, including the U.S. Army Engineer Waterways Experiment Station (WES) and the U.S. Army Corps of Engineers Huntsville Division (HND), is not only a problem for those sites, but is also an indicator that CEFMS performance could be a problem for other sites as deployment proceeds. To cope with this slowness, WES has been forced to allocate different time slots during the work day to different laboratories in order to reduce the demands on CEFMS. By all reasonable gauges of performance of an on-line, interactive system, the poor performance experienced to date and the means that must be employed to cope with that poor performance are unacceptable.

This problem has not been ignored. The CEFMS development team worked to improve CEFMS performance, and significant improvements were made during the fourth quarter of 1994 and the first quarter of 1995. The Corps of Engineers Automation Plan (CEAP) Program Manager assembled a team in the fall of 1994 to investigate how performance might be improved. That team made

[1] UNIX is a trademark of X/Open.

numerous recommendations for tuning both the program and the database structure; many of these recommendations will have been implemented by the time this report is issued. At this writing,[1] a special file system from Veritas is also being evaluated as a potential performance enhancer.

Improving the performance of CEFMS by installing a more powerful computer was also considered. However, the risk of either (a) purchasing a system that was too small and obtaining little or no improvement in performance or (b) purchasing a more powerful (and presumably more expensive) system when a smaller system would have done the job was unacceptable. A reliable way to predict the effects of hardware and software changes, and an effective way to measure the effects after changes were made, had to be obtained.

Conventions Used in this Report[2]

The following conventions are used in this report:

Bold	is used for statements and functions, identifiers, and program names.
Italic	is used for file and directory names when they appear in the body of a paragraph as well as for data types and to emphasize new terms and concepts when they are introduced. It is also used for titles of books and journals.
Constant Width	is used in examples to show the contents of files or the output from commands.
Constant Bold	is used in examples to show command lines and options that should be typed literally by the user.
Quotes	are used to identify a code fragment in explanatory text. System messages, signs, symbols, and quotations from other sources are quoted as well.
$	is the Bourne Shell prompt.
%	is the C Shell prompt.
[ ]	surrounds optional elements in a description of program syntax. (The brackets themselves should never be typed.)

[1] July 1997.

[2] Reprinted with permission from *sed & awk*, 2nd ed. © 1997 O'Reilly & Associates, Inc. (Dougherty and Robbins 1997). For orders and information call 1-800-998-9938.

Benchmarking Computer Systems

Benchmarking may be defined as a means of estimating the performance of systems by imposing one or more test workloads on them and then measuring their performance (Jones 1975). This broad definition may be applied in many contexts, but computers are the systems of particular interest here. Even with this restriction there is considerable latitude for the application of benchmarking; central processing units (CPUs), memory subsystems, input/output (I/O) subsystems, graphics subsystems, disk subsystems, compilers, operating systems, entire computer systems, multiprocessor computer systems, and local area networks all may be, and have been, benchmarked. The factor which generally serves to discriminate between benchmark techniques is not so much what is being tested, but the method each technique uses to create a representative test.

There are several aspects to the concept of benchmark representativeness. First, the test workload must be an accurate model of the projected actual workload. A hybrid workload consisting of a limited number of actual jobs may be more effective in this respect than a synthetic workload (Ferrari 1972). Use of programs still in the developmental stage may also be a relevant way to reflect future processing requirements in a benchmark test (BT) (Dongarra, Martin, and Worlton 1987). A second aspect of representativeness depends on the system configured to run the test; it should match the proposed configuration to the greatest extent possible. However, proposed systems with hundreds of attached terminals or networked nodes may be impossible to duplicate. A third aspect of this objective is the benchmarking methodology. If one is purchasing a large-scale system to run a relational database management system which will support a large number of users, then running one or two test jobs, or even a few actual programs, would scarcely approximate the actual operating environment. Several example methodologies are presented in the following paragraphs in rough order of representativeness. (Although lack of representativeness may reduce the credibility of a BT's results, it does not necessarily imply lack of effectiveness in measuring system performance.)

Among the least representative benchmarks is the synthetic job, described by Kernighan and Hamilton (1973) as "a program which uses precisely specified amounts of computing resources, but which does no 'useful' work." Such jobs may be constructed to replicate the CPU and I/O requirements of a real job without regard to actual program features, or they may be constructed to represent a typical program written in a high-level language. If this latter approach is used, one first obtains the frequencies with which various language features occur. Ideally, these should be dynamic frequencies taken from program execution traces; when these are not available, static frequencies taken from source programs may be substituted. The frequencies so obtained are then used to construct a program that exhibits the same frequencies of occurrence for all features of interest. The program may accept an input parameter to vary its resource utilization, or the program may be invoked within a loop to obtain a measurable amount of work. The results may then be expressed in instructions per second or program completions per second.

An early example of this technique is the Buchholz benchmark, based on a master file/transaction file update and written in PL/I (Buchholz 1969). This program was parameterized so as to adapt to varying given frequencies of computation and I/O. A second example, and one of the most widely used synthetic programs, is the Whetstone benchmark (Curnow and Wichman 1976), apparently named after an Algol interpreter system (Randell and Russell 1960) which was subsequently modified to produced both static and dynamic frequency counts of instruction types from a number of real programs. The program itself was constructed using 11 modules containing loops whose iteration counts were treated as variables; the values of these variables were then selected to allow the frequencies in the synthetic program to match the observed frequencies. A third example is the Dhrystone benchmark (Weicker 1984, 1988). Whereas Whetstone primarily measured floating-point performance, Dhrystone was characterized by its author as a "systems programming benchmark." To determine the frequency of individual high-level language constructs, 16 different data collections drawn from several languages were used. The constructs considered include types of operations, numbers of parameters, types of operands, operand locality, statement types, types of loops, and types of assignment statements. Although many synthetic benchmarks have been carefully constructed, they still have several shortcomings, as noted below.

a. Even if a synthetic program is perfectly representative in terms of language features and resource consumption, the workload it models is generic and will not necessarily match the workload on a given system.

b. Care must be taken in the construction of synthetic programs so that sophisticated compilers will not optimize all of the work out of the program. On the other hand, a program designed to completely defeat compiler optimizations is not representative.

c. Synthetic programs tend to be shorter than actual programs, and, unless multiple instances of the program are submitted simultaneously, it is quite possible they will become cache-resident and the results reported will be too optimistic. Recently announced microprocessor-based systems which include a 16 Kbyte on-chip data cache, a 20 Kbyte on-chip instruction cache, and a 1 Mbyte off-chip cache illustrate that this is a very real possibility.

A second methodology is the use of program *kernels*. To apply this technique, heavily-used programs from the actual workload are analyzed to determine which portions of the code use most of the computing time. These resource-intensive portions, or kernels, are extracted and combined into a single benchmark program. Examples include the NAS Kernel Benchmark Program (Bailey and Barton 1985), to be described in the following paragraphs, and the Livermore Fortran Kernels, commonly referred to as the *Livermore Loops* (McMahon 1986). Although the kernel approach appears to be somewhat more representative than the use of synthetic programs, there are several problems with its application:

a. Many kernel programs have been developed by research laboratories to model their vector supercomputer workloads. Unless the given workload happens to match the one upon which the benchmark was based, representativeness is lost.

b. Many kernels inherently favor a particular architecture. For example, if a benchmark is designed to measure vector supercomputer performance, then the source code modifications required to obtain enhanced performance on a large-scale multiprocessor might be extensive.

c. Care must be taken to preserve benchmark uniformity. According to Bailey and Barton (1985) "Some vendors have claimed amazingly high performance rates for their computers, which, upon closer analysis, have been achieved only by massive recoding of the test kernels and by the usage of assembly code." If modifications of this sort are performed, they should be clearly documented.

d. Kernel benchmarks generally measure processor/compiler performance; overall system performance is not addressed.

A third benchmarking methodology involves the use of one or more actual programs. Each member of the test suite is executed in a serial fashion (as opposed to a job stream), and a separate execution time for each is obtained. An average of these times may then serve as the performance rating for the system (other scores are possible). Several currently popular benchmarks use this methodology, including the PERFECT (Cybenko 1990,Grassl and Schwarzmeier 1990, Saavedra-Barrera 1990) and SLALOM (Gustafson et al. 1990) benchmarks. A third example is the SPEC benchmark suite (Saavedra-Barrera 1990).

The Standard Performance Evaluation Corporation (SPEC) is a nonprofit corporation formed to "establish, maintain and endorse a standardized set of relevant benchmarks that can be applied to the newest generation of high-performance computers" (quoted from SPEC's bylaws) (SPEC 1997). Although SPEC maintains several benchmarks, the two of interest here are the CINT95 and the CFP95 test suites. CINT95 contains eight C programs which perform integer computations; these programs include a Motorola 88000 chip simulator and a Lisp interpreter. CFP95 contains ten Fortran 77 programs which perform floating-point computations; these programs include a solver for Navier Stokes equations and simulation programs in quantum chemistry and physics. Each program is compiled in two modes, conservative optimization and aggressive optimization. Then, for each program/mode combination, a performance ratio of the system under test (SUT) execution time to the known Sun SPARCstation 10/40 80 execution time is calculated for each of the programs. Finally, geometric means of the conservative CINT95 ratios, the conservative CFP95 ratios, the agressive CINT95 ratios, and the agressive CFP95 ratios are computed. These four numbers, termed SPECint_base95, SPECfp_base95, SPECint95, and SPECfp95, respectively, are used to characterize the performance of the SUT/compiler being evaluated. SPEC periodically changes the programs used (the prior test suite was constructed in 1992) and also maintains a list of these four

values for various machines for public inspection on its World-Wide Web page (SPEC 1997). Although this approach is quite useful for relative comparisons of general computing capability of two systems, the use of actual programs in this fashion suffers from many of the same problems as the kernel approach, including lack of representativeness due to a fixed workload, possible lack of uniformity due to vendor modifications, and failure to measure overall system performance due to the way the workload is imposed on the system.

A fourth methodology requires the construction of a stream of jobs so that system throughput may be measured. Three types of drive workloads are possible: actual, artificial, and hybrid (Ferrari 1972). When the actual workload is used, measurements of system response must be taken to cover most, if not all, of the significant operating periods (e.g., end of the day, week, month, fiscal year, semester). Thus the measurement period is long, but little benchmark preparation is required. However, repeating a workload test so observed on a production system is impossible. Furthermore, when the purpose of benchmarking is for system selection, imposition of the actual workload on a system not yet installed is difficult.

Artificial workloads are composed of synthetic jobs and kernel programs. Such workloads typically involve shorter measurement periods, but more benchmark preparation. One example proposes the use of a synthetic stream (Wood and Forman 1971); this approach is useful when actual jobs are not available (e.g., when security and confidentiality are issues). Here, repeated instances of the Buchholz synthetic benchmark program noted above were used to build the stream. Parameter values controlling the amount of computation and I/O were chosen to correspond to actual jobs. Comparison with the use of an actual job stream verified the feasibility of this approach. An elaboration on the use of this idea involves the use of accounting data to provide job resource utilization characteristics (Sreenivasan and Kleinman 1974). In this case, the mix of jobs was chosen through a probability distribution technique with a limit on the total BT time. A second enhancement to this technique has also been implemented. It involves the automatic generation of a complete synthetic job, including job control statements, from input which specifies job resource utilization (Kernighan and Hamilton 1973).

In an attempt to improve representativeness, a hybrid workload, consisting of carefully selected actual jobs, may be used in place of the synthetic stream discussed above (Ferrari 1972). Here, the representativeness of the actual workload is preserved through the use of real jobs, while the shorter measurement period used in the artificial workload is possible because a job stream of similar size is used. Furthermore, benchmark preparation time is reduced because synthetic programs or kernels need not be prepared.

After the test workload has been constructed, another aspect to be considered is the method by which the workload is imposed on the SUT. One technique uses a program resident on the SUT to periodically submit jobs to the system; however, the presence of this additional program on the system biases the results. A better technique imposes the test workload on the SUT from an

external source. If the test workload involves a large number of interactive users, then actual configuration of the requisite number of terminals is not possible. In such circumstances, the terminals may be emulated and their input supplied by an external driver computer. This approach, to be discussed in the following section, is probably the most representative of all benchmarking methodologies.

Remote Terminal Emulation

According to the Federal Computer Performance Evaluation and Simulation Center (1979):

> Remote terminal emulation is one benchmarking technique for conducting tests of teleprocessing computer systems and services when it is impractical to configure for a test the total planned network of computers, teleprocessing devices, and data communication facilities. Remote terminal emulation uses an external driver computer and computer programs to imitate the teleprocessing devices to be supported by, and to impose the workload demands on, the actual computer system or service being tested (hereafter referred to as the System Under Test (SUT)). A remote terminal emulator (RTE) is a specific hardware and software implementation of this driver system. During acquisitions, each vendor provides and operates the RTE used for benchmarking that vendor's system. While any BT can be expensive, a BT using remote terminal emulation is usually costly and complex and can be technically invalid if improperly designed or conducted.

However, if the RTE is correctly configured, the SUT cannot distinguish whether the workload is imposed by an actual or an emulated population of remote terminals, and, if the workload is appropriately constructed, accurate system sizing is possible. It is important to realize that the terminal emulation in the sense discussed here is different from terminal emulation programs provided on personal computers (PCs). An RTE substitutes for a population of terminal sessions which could, as far as the SUT is concerned, be originating from direct-attached terminals or emulation programs on network-attached PCs.

A critical component of an RTE is a monitor that records in a log file every data transmission between the RTE and SUT along with a time-stamp indicating when the data was received by the RTE. After the test, this log file is summarized to produce various performance measures for the SUT (e.g., batch turnaround time and interactive response time). A complete description of the use and specifications of remote terminal emulation is given in the GSA handbook *Use and Specifications of Remote Terminal Emulation in ADP System Acquisitions* (FCPESC 1979). A vendor-independent portable RTE has been implemented (Adams, Currie, and Gilmour 1978), while a more recent, UNIX-based emulator which can communicate with the SUT over TCP/IP networks as

well as through asynchronous lines is available commercially (Pure Software 1996).

An RTE test workload is described using *scenarios*, sequences of one or more computing activities described in a vendor-independent fashion. The characteristics of the scenarios (e.g., batch or interactive, business or scientific, compile or execute) and the number of times each is repeated during the BT are collectively referred to as the benchmark mix. These generic scenarios are implemented in a particular computing environment using system-specific scripts written in the vendor's operating system control language, in a high-level programming language, or in a combination of both. For example, if an interactive scenario specifies the compilation and execution of a FORTRAN program and then prints all output lines containing "answer," then the corresponding script file on a UNIX system could contain the commands shown in Figure 1.

```
F77 -0 -time -o testprog main.f sub1.f sub2.f
testprog < input.dat > ouput.dat
grep answer output.dat
```

Figure 1. Simple example of a script

2 Test Workload Definition

Construction of a representative test workload was crucial to the credibility of these BTs. The first important decision made regarding this workload involved deciding what "representative" meant. Because the poor performance experienced by CEFMS users occurred during periods of high system utilization, it seemed appropriate to test worst-case performance. Accordingly, it was decided that each BT should represent the workload encountered during the worst 2-hr period of the worst processing day of the year. The following sections discuss how that peak workload was constructed.

Obtaining Transaction Counts

CEFMS user support personnel grouped CEFMS user tasks into functional areas and then identified specific activities within each area. Each activity was given an eight-character mnemonic name; the first three characters specified the functional area, and the remaining characters specified the activity. Table 1 gives the final list of activities represented in the test.

Each activity should have had a corresponding script in the BT. However, some activities, or some aspects of activities, proved difficult or impossible to implement using remote terminal emulation. The electronic signature verification (ESV) mechanism of CEFMS is implemented on a separate system from the CEFMS SUT. Capturing and emulating the ESV process with the given RTE tools proved to be impossible, so ESV was turned off during the test. Other candidate activities which required interaction with a system separate from the SUT were omitted from the test for the same reason; specifically, these were technical approval of a purchase request and commitment (PR&C) and logistics approval of a PR&C. Although this reduced the representativeness of the tests, much of the resource utilization for these activities is on systems external to the CEFMS production system. Therefore it is believed that the impact of their omission is not significant.

After specifying the candidate list of activities, the next step was to determine the frequency with which each was performed. When an activity modifies one or more database tables, those changes are logged by the database software. Because each activity modifies different tables, it is possible to examine the logs

Table 1
Script Names and Activities

Script Name	Activity Description[1]
coraccpt	accept a customer order
corcreat	create a customer order
corapprv	approve a customer order
corcertf	certify a customer order
cvocreat	create a collection voucher
cvocertf	certify a collection voucher
labcreat	create a labor PR&C
labctran	labor cost transfer
prccreat	create a PR&C
prcapprv	approve a PR&C
prccertf	certify a PR&C
prccreob	create an obligation for a PR&C
prcappob	approve an obligation for a PR&C
prccrein	create an invoice for a PR&C
repa3953	run a DoA 3953 report
repa4445	run a DoA 4445 report
repcertl	run a certlabor report
repcolds	run a coldsbrg report
repd1556	run a DoD 1556 report
repmscdb	run a misc. disbursement report
repsdipr	run a ? report
reptmatt	run a time & attendance report
reptrvdb	run a travel disbursement report
repvstat	run a Visa status report
torcreat	create travel orders
torreque	request travel orders
torapprv	approve travel orders
torauthn	authenticate travel orders
torcrevo	create a travel voucher
torappvo	approve a travel voucher
trncreat	create a training PR&C
viscreat	create a Visa PR&C
visapprv	approve a Visa PR&C

[1] PR&C = purchase request and commitment.

and determine how many times that activity was performed. The logs are stored in the database so an SQL script must be written to make the examination. An example of such a script is provided in Appendix A, and its output is presented in Figure 2. As shown in the figure, the output of each SQL script is a total by day of the number of times each activity was done; these are called *transaction counts*. The transaction count data are placed in separate files by activity. The names of these files are ultimately used as script names; a Bourne shell script, **cpdir**, was written to alias these filenames to the selected script names. This script is listed in Appendix A.

```
set pagesize 60
set echo off
set term off
set feedback off
set linesize 80
ttitle 'COUNT BY DAY OF THE PRAC_CERT.LST DURING TEST PERIOD'
column trunc(a.cert_date) heading 'DATE' format a10
column count(b.prac_line_no) heading 'COUNT' format 999999
spool prac_cert.lst
select trunc(a.cert_date), count(b.prac_line_no)
from   pr_amend a, pr_line-item b
where  a.cert_date is not null
and    b.certified_us_amt is not null
and    trunc(cert_date) between '01-SEP-95' and '30-SEP-95'
and    a.prac_no = b.prac_no
group by trunc(a.cert_date)
/
spool off
exit;
```

Figure 2. SQL script to count PR&C certifications

However, there is a significant problem with this approach. Activities that do not modify the database, such as queries and reports, will not be detected. In the case of queries, not only is the total number unknown, but also the fields being queried and, therefore, the relative field query frequency. Because of these issues, queries were not represented in the BT. The approach used to determine the frequency of reports is different from other activities and will be discussed in a subsequent section.

Producing Script Counts from Transaction Counts

As stated earlier, the raw data available for workload definition was in the form of files, one per activity, containing transaction counts. Each line in one of these files contained a date and a count. The data covered 1-30 September 1995. Given these data as input, the objective was to produce a list in which each line contained a script name followed by a script count. The script names were the same as the activity names; the implementation of each activity as an executable script is discussed in a later section. The script count specified how many times each script would be executed during the BT. The script count represents the number of times the corresponding activity would be performed during the worst 2-hr period of the worst day of the year.

The problem immediately encountered was to define *worst day*. The easiest approach would have been to select the day from the raw data with the greatest total number of transactions. However, on that day some activities were not performed at all while others were performed an unusually large number of times. While such a day may be termed *worst*, a BT with script counts based on that day's data would not be very representative. Another possibility would be to select the maximum count for each individual activity to construct a hypothetical

worst day. It is highly improbable, however, that the maxima for each activity would occur on the same day, so this approach overestimates the script counts.

The method finally adopted was statistical in nature. An average number of transactions per day and a standard deviation were calculated for each activity; weekend days and holidays (in this case, Labor Day 1995) were excluded from the calculation. A worst day was then defined as a day for which the transaction counts for each activity were one standard deviation (1SD) above the mean. (Interestingly, the total transaction count by this method was surprisingly close to that of the day with the greatest total number of transactions.) These 1SD transaction counts were then divided by 24 (hours) and multiplied by 2 (hours) to give a set of script counts which represented the number of transactions performed in an average 2-hr period on the worst day of the year. To obtain a BT workload representing the worst 2-hr period on the worst day of the year, these script counts were then multiplied by a *peaking factor*, the calculation of which is discussed in a subsequent section.

Some final adjustments had to be made to make the script counts consistent with the way activities were actually performed. For example, the transaction counts for PR&Cs were actually counts of line items. A tally of the PR&C documents revealed that, on the average, a CEFMS user enters 5.22 line items per PR&C. To emulate this situation, the **prccreat** script was designed to enter 5 line items, and the script count for **prccreat** was divided by 5 (a scale factor of 0.2). Similar compensations were implemented for other PR&C-related scripts. A more representative approach would have been to design the script to enter a variable number of line items and force the average line item count to equal 5.22. However, since considerable time would have been required to implement this approach, this compromise seemed justified. A similar approach was used to adjust the script counts for **labcreat** scripts to more accurately model the process of creating labor PR&Cs.

A Bourne shell script, **tc2sc**, was written to convert transaction counts to script counts; it has options to allow the use of peaking factors, the use of per activity scale factors as discussed above, and the exclusion of specific days from the computation, as discussed above. This script is listed in Appendix A.

Producing Script Counts for Report Scripts

As noted earlier, because reports do not modify the database, there are no transactions to log, and therefore no transaction counts. Fortunately, however, the reports themselves are directed to files prior to printing; these report files are given mnemonic names which indicate the type of report. UNIX gives every file a creation/modification time stamp and CEFMS retains a report file from several days to several months, depending on the user's preference. Using the UNIX command `ls -l`, it is possible to extract information on the type, creation date, and size of various reports which is reliable for the past several days. Unfortunately, the availability of information this detailed revealed another problem:

there are dozens of different types of reports, and capturing a terminal session for each was not feasible. Instead, reports were categorized by size into small, medium, and large, and the number in each category was tabulated. A few reports representative of these categories were selected for inclusion in the BT.

Determining the Peaking Factor

After the raw transaction count, data were analyzed to produce the average 2-hr workload on the worst day; this workload had to be scaled by a peaking factor to reflect the worst 2-hr period on that day. Obviously, the peaking factor could not be determined from the transaction counts because daily data were too coarse. Fortunately, Solaris (Sun's version of UNIX) has accounting procedures which, when activated, automatically accumulate process-related data in */var/adm/pacct* which is accurate to the second, and user login-related data in */var/adm/wtmpx* which is accurate to the minute.

The *pacct* file is a binary data file which contains one record for each process executed by the system. Each record contains, among other things, the process name, the name of the user who initiated the process, the start time and end time of the process (accurate to the second), the CPU seconds used by the process (accurate to hundredths of a second), and the average amount of memory used by the process. Further information on this file may be obtained by using `man -s 4 acct` or by inspecting the C library header file */usr/include/sys/acct.h*. The utility program **acctcom** is used to prepare a user-readable report from the *pacct* file. The first several lines of one such report are shown in Figure 3.

```
mailx       u4rmsslk  pts/6  19:02:48  19:02:48   0.07  0.07  1369.14
sh          u4rmsslk  pts/6  19:02:48  19:02:48   0.12  0.05  1620.80
#sendmail   u4rmsslk  pts/6  19:02:48  19:02:48   0.18  0.17  2165.65
#mail.loc   u4rmsslk  ?      19:02:48  19:02:48   0.09  0.08  1591.00
#sendmail   u4rmsslk  pts/6  19:02:48  19:02:48   0.16  0.06  2242.67
mailx       u4rmsslk  pts/6  19:03:17  19:03:17   0.08  0.08  1155.00
sh          u4rmsslk  pts/6  19:03:17  19:03:17   0.12  0.04  1578.00
#sendmail   u4rmsslk  pts/6  19:03:17  19:03:17   0.19  0.19  2091.37
#mail.loc   u4rmsslk  ?      19:03:18  19:03:18   0.11  0.10  1760.00
#sendmail   u4rmsslk  pts/6  19:03:18  19:03:18   0.18  0.07  2187.43
mailx       u4gvbame  ?      19:03:27  19:03:27   0.07  0.07  1284.57
sh          u4gvbame  ?      19:03:27  19:03:27   0.13  0.05  3318.40
#sendmail   u4gvbame  ?      19:03:27  19:03:27   0.18  0.17  2095.53
#mail.loc   u4gvbame  ?      19:03:27  19:03:27   0.11  0.09  1522.67
#sendmail   u4gvbame  ?      19:03:27  19:03:27   0.18  0.07  2197.71
mailx       u4rmsslk  pts/6  19:03:38  19:03:38   0.08  0.08  1213.00
sh          u4rmsslk  pts/6  19:03:37  19:03:37   0.12  0.04  1602.00
#sendmail   u4rmsslk  pts/6  19:03:38  19:03:38   0.18  0.18  2079.11
#mail.loc   u4rmsslk  ?      19:03:38  19:03:38   0.10  0.09  1639.11
#sendmail   u4rmsslk  pts/6  19:03:38  19:03:38   0.17  0.06  2293.33
```

Figure 3. Fragment of a *pacct* file after processing by **acctcom**

From the *pacct* data it is possible to construct a (virtual) graph of CPU time versus time of day. The area under some segment of this graph is the CPU time consumed during that time interval. Dividing the area by the length of the time interval gives the average CPU time used per unit time. On a uniprocessor this average must be ≤1; on a multiprocessor it must be ≤ the number of processors. Equality is only possible at 100 percent processor utilization. Multiplying this average by 2 hr, for example, gives the average CPU time used per 2-hr period. From the *pacct* data, it is also possible to determine the 2-hr period with the maximum consumption of CPU time. Dividing this maximum CPU time used over a 2-hr period by the average CPU time used over a 2-hr period gives a *peaking factor*.

Similarly, the *wtmpx* file contains one record for each user login. Each record contains, among other things, the login name, the date, the login time (accurate to the minute), and the logout time (accurate to the minute). Further information on this file may be obtained by using `man -s 4 utmp` or by inspecting the C library header file */usr/include/sys/utmp.h*. The utility program **last** is used to prepare a user-readable report from the *wtmpx* file. The first several lines of one such report are shown in Figure 4. From these data it is possible to construct a graph of number of users versus time of day and to calculate a peaking factor in the same manner as described above. Changes in the number of users very crudely approximate changes in the workload, so this peaking factor is not as useful for scaling an average workload to a worst-case workload. Nevertheless, this number-of-users peaking factor serves two purposes: first, it may provide supplemental evidence that the CPU time peaking factor is accurate, and second, it may be used to determine the number of emulated users to use in the BT.

```
u3plbmpk     pts/73   moonkim.cecer.ar  Thu    Sep 26    16:20 - 16:22 (00:01)
u3ul9gws     pts/52   moshage.cecer.ar  Thu    Sep 26    16:20 - 16:24 (00:03)
u4rmbgft     pts/65   134.164.76.5      Thu    Sep 26    16:20 - 16:27 (00:06)
u3plxkjc     pts/46   castle.cecer.arm  Thu    Sep 26    16:20  still logged in
s0rmfbda     pts/48   rmf22.hnd.usace.  Thu    Sep 26    16:19 - 16:21 (00:01)
u3plxkjc     pts/46   castle.cecer.arm  Thu    Sep 26    16:19 - 16:20 (00:00)
u4svbalk     pts/22   134.164.60.110    Thu    Sep 26    16:19 - 18:12 (01:52)
u4cvbbjs     pts/31   134.164.156.229   Thu    Sep 26    16:19 - 18:46 (02:27)
u3plxeec     pts/37   echris.cecer.arm  Thu    Sep 26    16:19 - 16:23 (00:04)
u4immewc     pts/25   134.164.40.168    Thu    Sep 26    16:19 - 16:21 (00:02)
u4imcljm     pts/20   134.164.72.21     Thu    Sep 26    16:19 - 16:25 (00:06)
u4eraeat     pts/36   elmer.wes.army.m  Thu    Sep 26    16:17 - 16:21 (00:03)
u4gpqslf     pts/24   134.164.180.161   Thu    Sep 26    16:17 - 17:36 (01:18)
u4imcfms     pts/10   cpc22             Thu    Sep 26    16:17  still logged in
u4er9clk     pts/25   134.164.100.217   Thu    Sep 26    16:16 - 16:18 (00:02)
u3llnkar     pts/35   reinbold.cecer.a  Thu    Sep 26    16:16 - 16:22 (00:05)
u3trwbwj     ftp      james.cecer.army  Thu    Sep 26    16:15 - 16:15 (00:00)
u3trwbwj     pts/10   james.cecer.army  Thu    Sep 26    16:15 - 16:16 (00:01)
c21mxcdb     pts/10   mrolm_12792.mro.  Thu    Sep 26    16:13 - 16:15 (00:01)
u3ctcdja     pts/28   adamson.cecer.ar  Thu    Sep 26    16:13 - 16:33 (00:19)
```

Figure 4. Fragment of a *wtmpx* file after processing by **last**

Bourne shell scripts **pa2pf** and **wt2pf** were written to calculate peaking factors from *pacct* and *wtmpx* files, respectively. Both of these scripts are listed in Appendix A. Care was taken to use **grep** to filter the input data to these scripts so that only records for WES users (those with login names beginning with "u4") were used to calculate peaking factors for a WES BT.

3 Mix Preparation

After all scripts have been successfully compiled and tested, and after the user has determined the number of times each script will be executed during the benchmark, the actual job mix must be prepared in a format acceptable to PurePerformix/TTY. The command actually used to initiate a BT is **mix**. **mix** requires two types of input; the first is a file, the *mix table*, that specifies which scripts will be executed in the BT and in what order, and the second is the file of *mix commandsfP* that initializes global variables and starts and resumes emulated users. Both of these inputs will be discussed in further detail in this section.

Producing the Mix Table from Script Counts

A PurePerformix/TTY mix table lists the emulated users along with the names of the scripts that each emulated user must perform during the test. A single emulated user may execute scripts which login under different UNIX user-names. The names of emulated users are known only to the RTE while the user-names are significant only to UNIX. A fragment of a mix table used in these tests is shown in Figure 5. Each line identifies a script to be executed during the test. The first field of each line is the name of an emulated user who will run the script, or a "+" to indicate that the previous emulated user should be reused. The second field is the script name; it must be the name of an executable file on the UNIX search path. The third field specifies the communications port with the SUT. Here, `telnet`, which indicates the mode of comunication (as opposed to a direct terminal line), is followed by the host Internet Protocol (IP) address or symbolic host name of the SUT. The name of the log file is the fourth field. Following fields contain arguments significant to the creator of the script. For these tests, the fifth field is the UNIX username, while the sixth and seventh fields are the script's input and output files, respectively.

The production of the mix table from the script counts is governed by several requirements. First, the number of emulated users in the mix table, which equals the number of users actually signed on to the SUT, should represent actual worst-case CEFMS usage. This worst-case number of users is obtained from the peaking factor calculation based on the *wtmpx* file of user login times discussed previously. Note that specification of the number of users (from the peaking factor calculation) and the number of scripts (from the script counts) determines

```
u0019,   cvocertf   telnet:cpc25   cvocertf0010   u4rmsddw      cvocertf0010
+        prccreat   telnet:cpc25   prccreat0015   u4imcbjc      prccreat0015
+        prccreob   telnet:cpc25   prccreob0023   u4lmscns      prccreob0023
+        prccrerr   telnet:cpc25   prccrerr0004   u4imbepg      prccrerr0004
+        visapprv   telnet:cpc25   visapprv0011   u4gsrjsh      visapprv0011
+        torauthn   telnet:cpc25   torauthn0009   u4rmfvlr      torauthn0009
+        prccreob   telnet:cpc25   prccreob0024   u4cvbdaw      prccreob0024
+        cvocertf   telnet:cpc25   cvocertf0011   u4rmfeem      cvocertf0011
+        repa3953   telnet:cpc25   repa39530004   u4rmfgdj      repa39530004
+        prccreat   telnet:cpc25   prccreat0016   u4espjmb      prccreat0016
+        prccreob   telnet:cpc25   prccreob0025   u4rmfdcg      prccreob0025
+        prccrein   telnet:cpc25   prccrein0037   u4rmsslk      prccrein0037
```

Figure 5. Sample mix table file used by **mix**

the number of scripts per user. Second, not every CEFMS user has the requisite CEFMS privileges to perform every CEFMS activity. This situation is handled by means of a script-user file which lists each script followed by the usernames allowed to run that script. Third, the initiation times of scripts should reflect the behavior of actual users (i.e., a mix table which schedules scripts of the same type at the same time is probably unrepresentative of actual CEFMS usage). Detailed data describing (a) which CEFMS users perform which tasks and (b) the ordering of tasks during a workday were unavailable. As a result, scripts were randomly assigned to emulated users and the order of script initiation was also randomized.

A Bourne shell script, **sc2mt**, was written to convert script counts to a mix table; it has options to specify the number of emulated users in the table and the name of the script-user file which handles the problem of CEFMS privileges. Randomization, which resolves the third issue noted above, is built into the shell script. This script is listed in Appendix A.

Producing the Preparatory Mix Table from the Mix Table

Some CEFMS activities may be performed only after one or more prior activities have been completed. For example, a PR&C may be certified only after it has been created and approved. In terms of executable scripts applied to a particular PR&C, the correct execution order is **prccreat**, **prcapprv**, **prccertf**, **prccreob**, **prcappob**, **prccrerr**, **prccrein**. Similarly, a travel order may be authenticated only after the travel order has been created, requested, and approved; the correct order in this case is **torcreat**, **torreque**, **torapprv**, **torauthn**, **torcrevo**, and **torappvo**.

There are several problems with including sequences of this sort in a BT. First and foremost, it is not representative; an individual PR&C or travel order is rarely, if ever, pushed through the system in a single day, much less in a single 2-hr period. Second, if script initiation times are randomly selected, it is difficult

to ensure that the requisite previous scripts have completed before the next script in the sequence is initiated without assigning the entire sequence to a single emulated user. This has the unrepresentative effect of forcing the initial scripts in the sequence to the beginning of the BT and the final scripts in the sequence to the end (e.g., the **prccreat** scripts would be executed relatively early in the BT). Third, in some cases, including an entire sequence is impossible. Specifically, travel orders must be issued before a trip begins, and travel vouchers, which handle employee reimbursement, must be issued after a trip is completed; CEFMS has built-in controls to prevent issuing a travel order and a travel voucher for the same trip on the same day. Therefore, **torcrevo**, which creates a travel voucher, may be executed, at the earliest, one business day after the corresponding **torauthn**.

The solution to this problem is to prepare the CEFMS database so that all the requisite prior transactions have been posted to the database prior to the actual BT. The number of prior transactions is large enough so that initiating these transactions manually is infeasible. Instead, a Bourne shell script, **mt2mt0**, creates a *preparatory mix table* that specifies all the necessary preparatory transactions. This shell script examines each script/line in the actual mix table to determine if it requires preparatory transactions. If it does, the necessary script/line(s) are written to the preparatory mix table. As an example, assume that the file fragment shown in Figure 5 is the actual mix table provided as input to this shell script. Then the output (the preparatory mix table) is given in Figure 6. The shell script itself is listed in Appendix A.

Producing Mix Command Files from Mix Tables

Both the actual mix table and the preparatory mix table must have an associated mix command file. This file contains commands to select the correct mix table, initialize global variables, set the time interval between user initiations, actually start the users, set the time interval between resuming successive suspended users, actually resume the users, and actually terminate the BT.

Users may be initiated using the mix command `start all`, or they may be initiated individually by name (using, for example, `start user003`), or they may be initiated at a specified time since the start of the test (using, for example, `at 120 start user003`). It was originally thought that the capability of starting particular users at particular times would be heavily used in order to maintain the desired load on the SUT. This turned out not to be the case in practice. However, it is still possible that some types of BTs may require this feature; this possibility is explored in a later section. In any case, the Bourne shell script **mt2mc**, which writes the mix command file, requires the name of the associated mix table file as input. This shell script is listed in Appendix A.

```
u0019, cvocreat   telnet:cpc25 cvocreat0010 u4rmfdcg  cvocreat0010  cvocertf0010
+      prccreat   telnet:cpc25 prccreat0116 u4imbmvs  prccreat0116  prccreob0023
+      prcapprv   telnet:cpc25 prcapprv0101 u4immhmh  prccreob0023
+      prccertf   telnet:cpc25 prccertf0080 u4rmsddw  prccreob0023
+      prccreat   telnet:cpc25 prccreat0117 u4evajwk  prccreat0117  prccrerr0004
+      prcapprv   telnet:cpc25 prcapprv0102 u4enshha  prccrerr0004
+      prccertf   telnet:cpc25 prccertf0081 u4rmfdcg  prccrerr0004
+      prccreob   telnet:cpc25 prccreob0058 u4imcwdp  prccrerr0004
+      prcappob   telnet:cpc25 prcappob0040 u4rmsslk  prccrerr0004
+      viscreat   telnet:cpc25 viscreat0011 u4gsrjsh  viscreat0011  visapprv0011
+      torcreat   telnet:cpc25 torcreat0018 u4gsrjsh  torcreat0018  torauthn0009
+      torreque   telnet:cpc25 torreque0013 u4hvcjph  torauthn0009
+      torapprv   telnet:cpc25 torapprv0009 u4hssbpf  torauthn0009
+      prccreat   telnet:cpc25 prccreat0118 u4er9clk  prccreat0118  prccreob0024
+      prcapprv   telnet:cpc25 prcapprv0103 u4imcbjc  prccreob0024
+      prccertf   telnet:cpc25 prccertf0082 u4rmsddw  prccreob0024
+      cvocreat   telnet:cpc25 cvocreat0011 u4rmsslk  cvocreat0011  cvocertf0011
+      prccreat   telnet:cpc25 prccreat0119 u4enrhrh  prccreat0119  prccreob0025
+      prcapprv   telnet:cpc25 prcapprv0104 u4eeaead  prccreob0025
+      prccertf   telnet:cpc25 prccertf0083 u4rmfals  prccreob0025
+      prccreat   telnet:cpc25 prccreat0120 u4hwrmjt  prccreat0120  prccrein0037
+      prcapprv   telnet:cpc25 prcapprv0105 u4hsljfg  prccrein0037
+      prccertf   telnet:cpc25 prccertf0084 u4rmfeem  prccrein0037
+      prccreob   telnet:cpc25 prccreob0059 u4pwzdrh  prccrein0037
+      prcappob   telnet:cpc25 prcappob0041 u4rmsslk  prccrein0037
```

Figure 6. Sample preparatory mix table file used by **mix**

4 Script Preparation

Understanding script preparation requires a knowledge of the RTE tools provided for this purpose; these are discussed in the first section of this chapter. The subsequent section describes how these tools were used to create prototype scripts and provides details on the customizations necessary to transform the prototypes into the scripts actually used in the BTs.

PurePerformix/TTY[1]

Version 3.2.2 of PurePerformix/TTY, an RTE licensed from Pure Software, Inc., was used to impose the workload on the various SUTs. This version of PurePerformix/TTY includes the following UNIX tools: **capture**, **compose**, **preview**, **scc**, **play**, **extract**, **report**, **mix**, and **draw**. The first five of these were frequently used to prepare scripts.

capture is used to create scripts from interactive sessions, with either the SUT or a compatible system. When keystrokes are entered, they are stored in a file before being sent to the specified communications port on the SUT. For example, entering **`capture prcappob /dev/tty4`** initiates an interactive session with the SUT via port */dev/tty4* on the RTE. File *prcappob.x* will contain a copy of every keystroke entered during the inter-active session. If **`capture prcappob telnet:cpc25`** is entered instead, then communication with SUT **`cpc25`** would take place using *telnet* rather than over a specified serial line. The *telnet* approach was used exclusively for the CEFMS tests. Additionally, when the **`-t`** option is specified, **capture** records type times and response times in the keystroke file for pos-sible later use by **compose**.

A sample keystroke file is shown in Figure 7. Lines in the keystroke file which do not begin with a tilde (~) are literal keystrokes typed by the user and transmitted to the SUT. Lines beginning with `~c` are comments. Lines beginning with `~t` contain the type time and response time noted above. Sequences such as `~K_ENTER` and `~K_F3` provide mnemonic representations of special

[1] The primary source for this section was Pure Software (1996).

```
~c TERM=vt220
~c capture -t -k vt220 prcappob telnet:localhost
~t 0.20 3.62
u4rmsslk~K_ENTER
~t 1.16 1.11
please~K_ENTER
~t 3.33 2.74
tms~K_ENTER
~t 26.98 4.37
3~K_ENTER
~t 1.23 1.17
5~K_ENTER
~t 0.99 0.94
5~K_ENTER
~t 1.94 1.89
6~K_ENTER
~t 7.51 7.42
~c CEFMS Enter Query
~K_F2
~t 7.17 7.17
w81ewf63196833
~t 0.20 4.77
~c CEFMS Execute Query
~K_F3
~t 30.44 30.12
y~K_ENTER
~t 3.87 3.83
~c VT220 End
~K_END
~t 15.73 14.92
~c CEFMS Exit or Previous Screen
~K_F10
~t 4.42 4.35
1~K_ENTER
~t 2.22 2.15
logout~K_ENTER
```

Figure 7. Sample keystroke file produced by **capture**

characters, in this case the enter key and the F3 function key, respectively. These mnemonic representations are defined in the *keys* file; use of this file is specified by the **-k** option to **capture** or by setting the shell environment variable **P_KEYS**. The *keys* file used to capture the scripts used in these tests is named *vt220.h*; it is listed in Appendix B.

After the keystrokes for a particular script are captured, the script itself must be constructed. This is done using the **compose** command; its output is a C source file, an example of which is shown in Figure 8. **compose** begins by writing the header portion of a C source file; this header is the initial 17 lines in the example. **compose** then reads the keystroke file created by **capture** and automatically retypes those keystrokes, thereby emulating an actual session with the SUT. As these keystrokes are sent to the SUT, `Xmit()` and `Kxmit()` statements are written to the C source file. When this C program is executed, those statements will transmit those same keystrokes to the SUT. As responses are received, `Rcv()` statements are written to the C source file. When the C

```
/* PurePerformix/TTY 3.2.2 */
/* TERM=vt220 */
/* capture -t -k vt220 prcappob telnet:localhost */
/* compose -a 10 -k vt220 prcappob telnet:localhost */

#include "vt220.h"

Set(CDELAY);                /* Put typing delay between characters */
Typerate(5);                /* Typing delay in CPS */
/* Bitrate(9600) ; */       /* Add RS-232 delays to response times */
Thinkuniform(1,2.5) ;       /* Think delay at every Xmit() */
Seed (getpid()) ;           /* Seed random number generator */
Timeout (300, CONTINUE) ;   /* What to do if Rcv()takes too long */
Unset (NOTIFY) ;            /* Don't display warnings.  Use Mon to find them */

/* Modify to disable Monitor Zoom or to change terminal descritpion  */
Term (ZOOM, VT220|LINES24|AUTOWRAP) ;

/* M^J^M^JUNIX(r) System V Release 4.0 (cpc25)^M^J^M^@^M^J^M^@login:      */
Rcv("^@^M^J^M^@login: " );

Kxmit("u4rmsslk", K_ENTER);
/* u4rmsslk^M^JPassword: */
Rcv("^JPassword: ");

Kxmit ("please", K_ENTER);
/*                              ^M^J^M^Jcpc25:/wes/u4rmsslk {33}%         */
Rcv("rmsslk{33}% ");

/* Suspend(); */

Beginscenario ("prcappob") ;

Kxmit ("tms", K_ENTER);
/* qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqj^ [[24;1H^OCount: *0^[[17;51H    */
Rcv("t: *0^[[17;51H");

Kxmit("3", K_ENTER);
/* e enter selection:     ^[[47;30;22m2^[[17;57H^[[0;44;37;1m ^[[17;51H   */
Rcv(";1m ^[[17;50H");

Ksmit("5", K_ENTER);
/* Please enter selection:   ^[[47;30;22m2     ^[[18;51H^[[0;44;37;1m     */
Rcv("  ^[[18;51H^[[0;44;37;1m");
```

Figure 8. Sample script produced by **compose** (Continued)

program is executed, those statements will cause the program to wait until the specified string is received from the SUT.

Of particular interest is the technique whereby **compose** detects the end of a response from the SUT. Although it is possible to instruct **compose** to wait for a period of silence on the communication line as a signal that the SUT response is over, this alternative was not chosen. A heavy workload may increase response times and cause **compose** to interpret a delayed response as the end of a

```
Kxmit("5", K_ENTER);
/*  Hplease enter selection:   ^[[47;30;22m2     ^[[19;47H^[[0;44;37;1m      */
Rcv("    ^[[19;47H^[[0;44;37;1m");

Kxmit("6", K_ENTER);
/*  Y to accept or 'D' to delete. Then <ENTER>.^[[ 2;18H^[[0;44;37;1m^G      */
Rcv("0;44;37;1m^G");

/*  CEFMS Enter Query  */
Kxmit ("", K_F2);
/*                        ^ [ [ 24 ; 35H^[[0;44;37;1mENTER QUERY^[[3;18H    */
Rcv("ENTER QUERY^[[3;18H");

Xmit("w81ewf63196833");
/*  8^[[0;44;37;1m^[[47;30;22m3^[[0;44;37;1m^[[47;30;22m3^[[0;44;37;1m       */
Rcv("[47;30;22m3^[[0;44;37;1m^[[47;30;22m3^[[0;44;37;1m");

/*  CEFMS Execute Query  */
Kxmit("", K_F3);
/* cept or 'D' to delete. Then <ENTER>.^[[24;9H^[[0;44;37;1m1^[[2;18H        */
Rcv("40;37;1m1^[[2;18H");

Kxmit("y", K_ENTER);
/* COMMIT>  TO PROCESS .                    ^[[2;18H^[[0;44;37;1m^G          */
Rcv("0;44;37;1m^G");

/*  VT220 End  */
Kxmit("",K_END);
/*  complete - - 6 records posted and committed.^[[2;18H^[[0;44;37;1m^G      */
Rcv("0;44;37;1m^G');

/*  CEFMS Exit or Previous Screen */
Kxmit("", K_F10);
/*                                          ^[[24;9H^[[0;44;37;1m0^[[19;47H  */
Rcv("7;1m0^[[19;47H");

Kxmit("1", K_ENTER);
/*  ;1m^[[2J^O^[[0;44;37;1m^[[?11^[>^[[m^[[2Jcpc25:/wes/u4rmsslk{34}%         */
Rcv("msslk{34}% ");

Kxmit("logout", K_ENTER);
/* logout^M^J   ^M^J  You have now logged off cpc25 . . . .    ^M^J ^M^J      */
Wait(2);

Endscenario("prcappob");
```

Figure 8. (Concluded)

response; the solution is to make the specified period of silence longer than any conceivable response time delay. Unfortunately, this approach results in artificial delays being present in the script. To avoid these problems, **compose** examines the keystroke file to determine a unique character string which will serve as a pattern to mark the end of a response.

Scripts, like most other programs, must be tested, debugged, and modified. The **preview** command facilitates this process by interpreting scripts produced by **compose**. This type of execution is not recommended for actual benchmark

testing because of its slowness relative to the execution of a compiled program, but it is ideal for use during script development. **preview** can only interpret PurePerformix/TTY commands (e.g., those generated by **compose**); if C language statements have been inserted in the script, they are ignored. The script must be compiled for these statements to take effect. Like a compiled script, **preview** produces a log file that contains a time-stamped entry for each transmission by the RTE and each response from the SUT. Additionally, by specifying the **-d** option, the emulated session will be displayed on the screen as it takes place.

When the user is satisfied with the script, the script C compiler **scc** is used to compile it. **scc** preprocesses the script file, adding, among other things, an appropriate header and trailer to make the script a valid C program. After this step, it then invokes the actual C compiler **cc** to produce an object program and to link that program with a special library of PurePerformix/TTY functions. As noted above, execution of the resulting program is faster than script interpretation using **preview**.

As an example, assume *prcappob.c* is the script produced by **compose**. Then `scc prcappob` will compile *prcappob.c* and produce an executable program named **prcappob**. Entering **`prcappob -d telnet:cpc25 prcappob.log2`** executes the script; the **`-d`** option causes the emulated session to be displayed, **`telnet:cpc25`** instructs the script to connect to host *cpc25* using **telnet**, and *prcappob.log2* will contain all of the responses transmitted by the SUT to the RTE, as well as transaction and response times. If this latter parameter is omitted, times will be logged in *prcappob.l:* if "" is specified, then no log file will be produced. Other script-specific parameters, such as login names and passwords, may be supplied after these. After **prcappob** has been executed and a log file has been produced, **`play prcappob`** may be used to replay the output contained in the log file. If **`play -s`** is used, then the user may step through the playback one response at a time by pressing the space bar. By default, **play** replays the session at the speed indicated by the time stamps in the log file. When this is too fast, the **`-f`** option may be used to slow down the replay.

Development of CEFMS Scripts

Before script development could begin, a snapshot of the WES production CEFMS database on *cpc25* was taken. This was necessary to ensure benchmark uniformity (i.e., every BT would impose the same tasks on the same database). Also, installation of this test database allowed BT development work to proceed without modifying actual production data. In this environment, single scripts or small batches of scripts could be tested, but a full-fledged BT (with 150+ emulated users) could not be attempted during normal production hours because of the negative impact on user response time.

An important issue involved how CEFMS activities were translated into scripts. Many CEFMS users log on to *cpc25*, perform one CEFMS task, and then log off. Others sign on in the morning, perform CEFMS tasks throughout their shift, and log off when their shift is over. Using PurePerformix/TTY, it is possible to construct scripts which model both types of user behavior. However, the second modeling mode carries some risk; if a script containing multiple tasks should hang or otherwise experience an error shortly after starting, then subsequent tasks may either execute improperly or not execute at all. Using the first mode avoids this difficulty since each script consists of a login-CEFMS task-logout sequence. Any failures are confined to that one script. To avoid this difficulty, the first mode was used. Although this diminished the representativeness of the tests, it allowed better timing of individual CEFMS activities and more rapid script development.

Script development was accomplished using the PurePerformix/TTY tools previously described. Actual CEFMS users were recruited to perform the activities noted earlier. They were signed on to *cpc25*, and their interactive responses were recorded using **capture**. The keystroke files so created were processed by **compose**, and then **scc** was used to produce C source files.

Because each scenario would be performed multiple times during a test, it was necessary to parameterize the scripts. Each script (i.e., each C source program) was modified so that the UNIX login name could be supplied from the command line as an argument. Some scripts required additional information, such as work-item numbers or PR&C numbers, to execute correctly. In other cases, preparatory scripts had to produce output information that was later used as input to subsequent scripts. To handle this situation, two addtional command line parameters were added to each script, an input file name and an output file name.

It was important that a number of activities performed by scripts be done the same way by every script. These activities included reading global variables,[1] setting RTE parameters, processing the argument list, reading the input file, logging into UNIX, entering CEFMS, suspending the script, leaving CEFMS, and logging out of UNIX. C header files were developed to perform these tasks; they were inserted into every script to ensure that the tasks were handled in a consistent manner. These header files are listed in Appendix B.

Finally, the scripts were individually tested; each was invoked from the command line on the RTE, causing the interactive session to be imposed on the SUT. This served two purposes: first, the scripts themselves were validated, and second, the processor time used by each was recorded.

---

[1] *Global variables*, a PurePerformix/TTY feature, were used to specify values common to every script; these were suspend status, think times, time-out duration, and type rate.

5 Benchmark Test Results and Conclusions

Summary of BT Methodology

To summarize the test methodology described to this point, preparing for a BT requires the following steps:

a. Determine the activities (transactions and reports) to be represented in the BT.

b. Use SQL scripts to determine the number of times these activities were performed over, for example, a one-month period.

c. Use the Bourne shell scripts discussed previously to convert activity counts to mix tables and mix command files.

d. Use **capture** and **compose** to create scripts for each of the activities.

e. Run a mix using the preparatory mix table to prepare the CEFMS database; save a copy of this prepared database.

f. If necessary, restore the prepared database.

g. Perform a BT by running a mix using the actual mix table.

h. If some scripts fail, fix the problem and back up the appropriate number of steps.

i. Use **extract** and **report** to produce reports on the elapsed script times and response times observed in the BT.

j. Save copies of the *pacct* and *wtmpx* files for later analysis.

Results of Three Benchmark Tests

On the evening of 4 May 1997, three BTs were performed. The RTE was a Sun file server (**wescs2.wes.army.mil**) with four 167-MHz UltraSPARC processors, while the SUT was a Sun file server (**cpc25.usace.army.mil**) with twenty-two 167 MHz UltraSPARC processors. The specifications of both of these systems are listed in Appendix C. These specifications include the layout of the CEFMS database on the SUT. The hardware configurations for all three BTs were identical except that, for BT3, half of the processors were taken off-line.

The uniformity of the software configurations was maintained to as great a degree as possible. The CEFMS database was restored prior to each BT. Think times were randomly distributed between 0.5 and 1.5 seconds, and the type rate was set to 5 characters per second. The same mix table was used throughout. Only one script failure occurred, a single instance of **labctran** on BT 2. The durations of the tests were 01:09:52, 01:10:21, and 01:10:19 for BTs 1, 2, and 3, respectively. This last time is noteworthy in that it indicates that the machine is not CPU-bound; more specifically, half the number of processors are apparently able to complete the workload in the same amount of time as the full complement. This could be due to the system's being bound by input/output or by think time. The BT was designed to be a 2-hr test based on data from a Sun file server (**cpc22.usace.army.mil**) with fourteen 60-MHz SuperSPARC processors. Thus, even though it is tempting to conclude that **cpc25**, which has over four times the total MHz of **cpc22**, is less than two times more powerful, that would be a conclusion not supported by the evidence. To measure the relative power of the two systems would require running the BT on **cpc22**. Furthermore, one such BT would still be insufficient in the statistical sense; enough BTs would have to be conducted to perform a hypothesis test.

A user workload curve for BT 1 is shown in Figure 9. It illustrates the tapering off of the number of active users, as opposed to a clean test termination; some manual adjustments were made to the mix table to prevent this phenomenon from being even more pronounced. The workload curves for the other two BTs were essentially identical. Additional detailed information on scenario completion is contained in Tables 2-7 which present scenario and interactive response time data.

Problems Encountered

When this project was initiated in 1995, the RTE hardware platform was a luggable Compaq 486 PC. The operating system on this PC was SCO Open Desktop 1.1 (UNIX System V/Version 3.2 with Berkeley enhancements). In spite of numerous attempts to tune the system, this RTE was inadequate to drive the SUT. Scripts terminated improperly because their remote connections were lost, the file system was unable to handle the large number of open log files, and other failures were encountered. Migrating to the current larger Sun platform

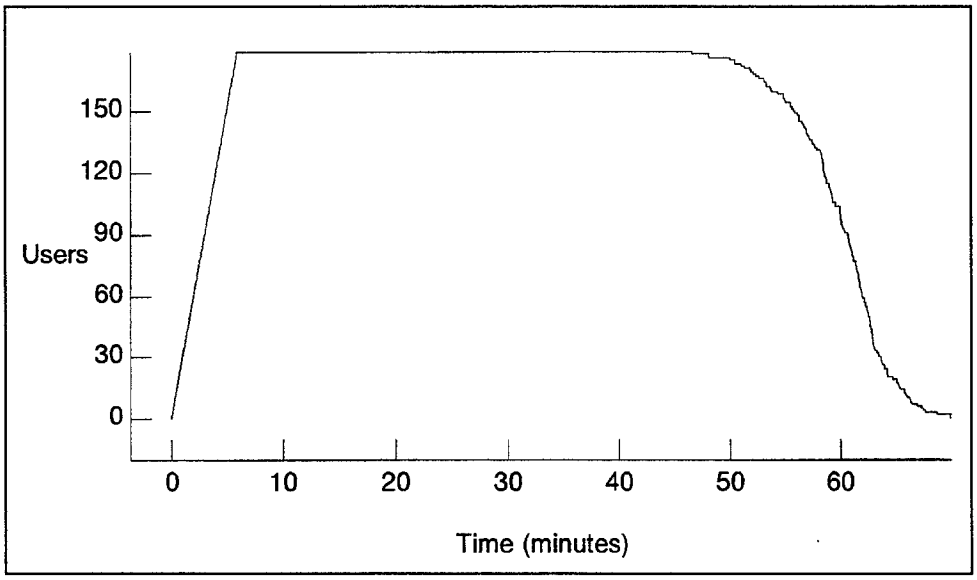


Figure 9. Active users during BT 1

Table 2
Scenario Statistics for BT 1

Scenario	Number of Times Executed	Average seconds	Std. Dev.	Min.	50th Percentile	75th Percentile	90th Percentile	Max.
coraccpt	12	241.07	3.86	235.97	240.10	242.49	246.99	247.31
corapprv	12	43.78	1.13	41.41	43.69	44.81	45.14	45.26
corcertf	12	44.28	1.10	42.92	44.14	44.97	45.39	46.73
corcreat	12	274.33	4.01	266.71	273.31	276.80	278.65	282.65
cvocertf	112	44.10	1.20	40.91	43.99	44.82	45.63	48.56
cvocreat	111	81.41	1.91	76.37	81.60	82.74	83.91	85.88
invcreat	297	67.59	2.00	62.55	67.50	68.82	70.19	75.27
labcreat	20	362.69	4.28	354.65	361.65	365.28	368.08	373.03
labctran	21	172.10	7.44	167.18	169.34	172.64	175.85	202.77
prcappob	169	43.32	1.19	39.89	43.24	44.07	44.56	50.22
prcapprv	169	47.32	1.37	43.98	47.19	48.08	48.77	57.47
prccertf	169	45.95	1.41	42.54	45.86	46.63	47.29	55.66
prccreat	169	283.00	4.00	274.55	282.91	285.93	288.10	294.38
prccreob	169	223.86	3.50	216.14	223.46	225.99	228.54	233.44
repa3953	23	91.79	3.93	86.92	91.35	93.32	94.67	106.72
repa4445	1	676.08	0.00	676.08	676.08	676.08	676.08	676.08
repcertl	23	108.33	2.41	103.46	108.41	109.60	111.39	113.14
repcolds	1	426.84	0.00	426.84	426.84	426.84	426.84	426.84
repmscdb	39	71.01	1.86	67.60	70.78	71.45	71.80	81.07
repsdipr	34	98.15	2.05	94.28	97.96	99.38	101.04	102.68
reptmatt	23	114.23	2.57	108.29	114.31	115.52	116.79	120.02
repvstat	1	893.83	0.16	893.83	893.83	893.83	893.83	893.83
rrecreat	54	54.44	1.70	51.62	54.27	55.39	57.18	59.41
torapprv	54	75.75	1.61	72.60	75.56	76.94	77.93	79.17
torauthn	38	63.02	1.31	60.98	62.85	63.64	64.94	66.33
torcreat	53	208.61	4.51	201.66	208.30	210.94	213.77	225.11
torreque	38	56.76	1.52	53.96	56.45	57.83	58.64	60.78
visapprv	89	45.42	1.15	42.57	45.26	46.19	47.08	48.08
viscreat	89	114.94	2.14	109.72	114.86	116.27	117.97	120.05
Overall	**2,014**	**105.83**	**85.89**	**39.89**	**67.47**	**115.24**	**271.72**	**893.83**

Table 3
Interactive Response Time Statistics for BT 1

Transaction	Number of Times Executed	Average seconds	Std. Dev.	Min.	50th Percentile	95th Percentile	99th Percentile	Max.
ctrl_f1	1,924	0.07	0.03	0.01	0.07	0.12	0.16	0.74
ctrl_f2	418	0.08	0.04	0.00	0.08	0.11	0.15	0.34
ctrl_f3	304	1.35	1.48	0.06	0.10	3.30	3.61	3.82
ctrl_f4	12	0.07	0.00	0.07	0.07	0.08	0.08	0.08
ctrl_f6	89	0.18	0.03	0.15	0.18	0.26	0.27	0.28
delete	67	0.00	0.00	0.00	0.00	0.00	0.00	0.00
down	7,546	0.00	0.01	0.00	0.00	0.01	0.04	0.10
end	4,525	0.61	2.98	0.00	0.12	1.33	6.98	42.74
enter	59,444	0.12	4.34	0.00	0.01	0.11	0.46	823.25
erase	717	0.00	0.01	0.00	0.00	0.01	0.01	0.10
f10	5,098	0.07	0.08	0.00	0.07	0.16	0.18	1.10
f2	2,528	0.03	0.02	0.00	0.01	0.06	0.07	0.18
f3	3,598	0.50	0.94	0.05	0.07	2.97	4.00	11.50
f4	7,338	0.18	0.26	0.01	0.07	0.63	1.31	3.00
f7	676	0.02	0.01	0.01	0.02	0.02	0.03	0.10
f9	1,417	0.02	0.02	0.00	0.03	0.05	0.08	0.23
other	9,572	0.01	0.11	0.00	0.00	0.02	0.10	7.93
page_down	371	0.07	0.04	0.00	0.09	0.10	0.13	0.16
page_up	1,352	0.05	0.02	0.02	0.06	0.07	0.08	0.15
shift_tab	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00
tab	1,322	0.01	0.01	00.0	0.00	0.04	0.05	0.10
tms	2,014	22.95	0.16	22.61	22.91	23.24	23.47	26.48
up	169	0.00	0.01	0.00	0.00	0.01	0.01	0.09
Unspecified	6,042	0.26	0.40	0.00	0.05	0.70	2.54	6.42
Overall	**116,544**	**0.53**	**4.34**	**0.00**	**0.02**	**0.63**	**22.91**	**823.25**

eliminated most of these difficulties. Future BT developers should be careful to select a machine powerful enough to drive a test.

During the initial BTs, it was discovered that when multiple PR&Cs were created simultaneously, duplicate PR&C numbers were generated by CEFMS; in computer science terminology, there was no locking of the critical region in the CEFMS code that produced the numbers. Some of the subsequent PR&C type scripts that depended on one of these initial prccreat scripts failed as a result. Workarounds for this problem were necessary until the error in the code was fixed.

A related problem involved the configuration management of the database software. Periodically, BTs that completed successfully one day would unexplainedly fail the next. It was discovered that the CEFMS development team was making nightly changes in the production version of CEFMS on a frequent basis. Occasionally, a source code modification fixing one problem would create another problem. The test coverage of the BT would often uncover the new bug. After several episodes of recapturing scripts, the versions of both the CEFMS database and software was frozen.

Table 4
Scenario Statistics for BT 2

Scenario	Number of Times Executed	Average seconds	Std. Dev.	Min.	50th Percentile	75th Percentile	90th Percentile	Max.
coraccpt	12	241.39	4.81	229.47	241.90	244.33	245.97	247.51
corapprv	12	44.11	1.10	41.78	44.31	44.70	45.43	45.68
corcertf	12	44.45	0.80	43.61	44.10	44.46	45.37	46.72
corcreat	12	274.47	3.41	268.55	274.04	274.71	278.20	282.76
cvocertf	112	44.31	1.20	41.31	44.28	45.10	45.95	47.39
cvocreat	111	82.12	2.08	77.88	82.08	83.86	85.01	85.97
invcreat	297	68.07	2.05	62.55	68.09	69.34	70.83	73.80
labcreat	20	363.15	4.53	356.45	362.15	366.57	368.84	372.78
labctran	20	175.62	9.05	165.50	173.80	177.54	180.61	210.52
prcappob	169	43.34	1.15	40.63	43.39	44.05	44.71	47.63
prcapprv	169	47.35	1.36	44.43	47.27	48.12	48.77	57.57
prccertf	169	46.00	1.52	43.09	45.98	46.88	47.53	55.90
prccreat	169	283.25	4.17	271.05	283.42	285.98	288.77	294.46
prccreob	169	224.66	3.37	214.82	224.80	227.21	228.84	233.96
repa3953	23	95.46	5.78	89.43	93.65	96.34	102.13	117.73
repa4445	1	716.11	0.00	716.11	716.11	716.11	716.11	716.11
repcertl	23	109.17	3.20	104.63	108.19	110.08	115.25	116.32
repcolds	1	468.15	0.07	468.15	468.15	468.15	468.15	468.15
repmscdb	39	71.06	2.10	68.86	70.90	71.41	72.28	82.29
repsdipr	34	98.70	2.49	94.24	98.11	99.95	101.33	106.71
reptmatt	23	114.64	3.50	109.81	114.38	115.08	118.47	126.94
repvstat	1	962.49	0.14	962.49	962.49	962.49	962.49	962.49
rrecreat	54	54.69	1.65	51.42	54.63	55.65	56.96	58.45
torapprv	54	76.51	1.85	72.22	76.15	77.65	79.52	80.66
torauthn	38	62.95	1.40	60.34	62.93	63.97	64.83	65.93
torcreat	53	209.66	3.83	202.04	209.50	212.77	214.80	217.20
torreque	38	57.14	1.36	53.52	56.95	58.12	59.05	59.38
visapprv	89	45.61	1.35	42.02	45.40	46.35	46.97	50.77
viscreat	89	115.29	2.34	109.20	115.12	116.87	118.84	121.10
Overall	**2,013**	**106.29**	**86.57**	**40.63**	**68.03**	**115.63**	**271.05**	**962.49**

An even more serious problem involved the UNIX usernames used in the test. These names were selected from actual WES CEFMS users. It was quickly discovered that certain tasks could be performed only by users with the right set of CEFMS permissions. Furthermore, there were critical relationships between users who "owned" particular work-item codes and other users who created PR&Cs. Enforcing these requirements was accomplished by creating a *script-user file* that specified which users could legally execute a script and a *user-work-item code file* that specified which work-item codes were owned by a particular user. The former file was an input to **sc2mt** and both were input to **mt2mt0**.

Future Work

A problem currently under study involves the shape of the *workload curve*. There are a number of possible measures of machine activity, including number

Table 5
Interactive Response Time Statistics for BT 2

Transaction	Number of Times Executed	Average seconds	Std. Dev.	Min.	50th Percentile	95th Percentile	99th Percentile	Max.
ctrl_f1	1,924	0.07	0.03	0.01	0.07	0.12	0.18	0.54
ctrl_f2	418	0.08	0.04	0.00	0.08	0.13	0.17	0.27
ctrl_f3	304	1.46	1.61	0.06	0.10	3.57	4.01	4.47
ctrl_f4	12	0.08	0.03	0.07	0.07	0.08	0.17	0.17
ctrl_f6	89	0.20	0.03	0.17	0.19	0.26	0.27	0.29
delete	67	0.00	0.00	0.00	0.00	0.01	0.01	0.01
down	7,546	0.01	0.01	0.00	0.00	0.01	0.01	0.11
end	4,526	0.64	3.02	0.00	0.14	1.52	9.57	42.76
enter	59,442	0.13	4.70	0.00	0.01	0.11	0.43	889.88
erase	717	0.00	0.01	0.00	0.00	0.01	0.02	0.10
f10	5,097	0.07	0.11	0.00	0.07	0.16	0.18	3.78
f2	2,528	0.03	0.03	0.00	0.01	0.06	0.06	0.45
f3	3,598	0.56	1.06	0.04	0.07	3.53	4.42	13.41
f4	7,338	0.18	0.28	0.01	0.07	0.68	1.42	4.21
f7	676	0.02	0.01	0.01	0.02	0.03	0.05	0.19
f9	1,417	0.02	0.02	0.00	0.03	0.05	0.06	0.19
other	9,572	0.01	0.11	0.00	0.00	0.02	0.11	7.98
page_down	371	0.07	0.04	0.00	0.09	0.10	0.13	0.37
page_up	1,352	0.05	0.02	0.02	0.06	0.07	0.08	0.16
shift_tab	1	0.01	0.00	0.01	0.01	0.01	0.01	0.01
tab	1,322	0.01	0.01	0.00	0.00	0.04	0.04	0.12
tms	2,014	22.96	0.16	22.67	22.92	23.23	23.41	26.19
up	169	0.00	0.01	0.00	0.00	0.01	0.01	0.03
Unspecified	6,041	0.25	0.39	0.00	0.05	0.70	1.03	3.14
Overall	**116,541**	**0.54**	**4.53**	**0.00**	**0.02**	**0.65**	**22.92**	**889.88**

of users and number of processes. A BT should ideally be a snapshot of machine activity with a sharp startup and a sharp cutoff. Currently, the start of the curve is rather steep, and its slope can be controlled (within limits) by setting a parameter that controls the interval between resuming suspended scripts. However, the end of the curve has a particularly severe tailing-off of lingering jobs. Ideally, all emulated users should complete their scripts at about the same time.

There are both static (mix-table-based) and dynamic (mix-command-based) approaches to solving this problem. A current approach to this problem involves running the mix to determine which RTE users are lingering, and then manually editing the mix table to move scripts from those users to RTE users that finish early. This approach could be automated, or the individual script execution times could be used to assign scripts to RTE users so that the total times are balanced among those users. A better approach would require devising a scheduling algorithm, perhaps involving assigning priorities to scripts and selecting the next script to execute as the BT progressed. This could be accomplished by modifying the mix table so that each RTE user had only one script, and then writing a program to dynamically interact with **mix** so that some fixed number of users was maintained on the SUT. It might be necessary to save such a dynamically created script initiation order so that the BT could be repeated.

Table 6
Scenario Statistics for BT 3

Scenario	Number of Times Executed	Average seconds	Std. Dev.	Min.	50th Percentile	75th Percentile	90th Percentile	Max.
coraccpt	12	242.66	3.19	238.03	243.03	243.70	245.97	248.81
corapprv	12	44.47	1.18	42.39	44.00	45.28	45.82	46.25
corcertf	12	45.22	1.25	42.22	45.22	45.59	46.78	47.47
corcreat	12	276.17	5.05	271.72	274.04	276.63	282.56	290.23
cvocertf	112	44.89	1.46	41.32	44.79	45.53	46.75	50.20
cvocreat	111	82.12	2.02	76.89	82.12	83.49	84.66	88.60
invcreat	297	68.43	2.41	62.86	68.20	69.91	71.22	76.94
labcreat	20	364.32	3.68	360.02	363.01	366.37	367.51	375.62
labctran	21	176.92	9.72	168.35	174.12	179.89	185.36	207.58
prcappob	169	43.72	1.29	40.88	43.67	44.52	45.20	49.33
prcapprv	169	47.83	1.59	44.73	47.54	48.52	49.79	55.27
prccertf	169	46.55	1.62	43.14	46.32	47.29	48.54	53.66
prccreat	169	284.25	4.21	274.63	284.21	287.51	289.38	294.14
prccreob	169	224.54	3.73	215.03	224.67	226.82	229.00	236.28
repa3953	23	99.44	8.01	92.32	96.21	99.04	110.73	122.04
repa4445	1	751.96	0.02	751.96	751.96	751.96	751.96	751.96
repcertl	23	110.22	4.14	103.88	109.51	110.55	115.90	121.65
repcolds	1	462.87	0.00	462.87	462.87	462.87	462.87	462.87
repmscdb	39	70.44	4.93	43.49	70.91	71.40	73.12	82.20
repsdipr	34	101.79	4.11	96.99	101.03	103.04	104.83	119.81
reptmatt	23	117.17	6.84	108.12	115.72	119.00	126.33	136.19
repvstat	1	947.52	0.01	947.52	947.52	947.52	947.52	947.52
rrecreat	54	54.91	1.87	51.47	54.70	56.23	57.06	61.27
torapprv	54	76.92	2.38	71.58	76.82	77.77	80.25	84.98
torauthn	38	63.46	2.01	59.32	63.38	64.50	64.93	70.95
torcreat	53	210.45	4.55	200.66	209.91	212.51	216.78	224.08
torreque	38	57.61	1.93	53.24	57.20	58.40	60.16	62.63
visapprv	89	45.82	1.23	42.99	45.77	46.54	47.12	49.82
viscreat	89	115.95	2.22	109.94	115.88	117.58	118.79	122.04
Overall	**2,014**	**106.88**	**86.73**	**40.88**	**68.12**	**117.08**	**273.15**	**947.52**

Internally developed tools also deserve additional attention. These include the shell scripts used to automatically construct script tables and batch command files for **mix**. Additional features, such as use of randomly distributed intervals between script initiations, and more options to control mix content should be considered. Also, some additional tools should be developed to summarize and compare the results of multiple benchmark tests. These results currently reside in a multiplicity of subdirectories, and items of interest must be individually extracted for inclusion in a summary report.

Another important area for further work requires making the mix and the scripts themselves more representative. The mix itself can be improved by basing it on more and better raw workload data. This will require collecting a larger set of *pacct* and *wtmpx* files, as well as monitoring the number and type of reports created. Furthermore, something must be done to obtain reliable data on the number of queries being performed.

Table 7
Interactive Response Time Statistics for BT 3

Transaction	Number of Times Executed	Average seconds	Std. Dev.	Min.	50th Percentile	95th Percentile	99th Percentile	Max.
ctrl_f1	1,924	0.08	0.03	0.01	0.07	0.13	0.17	0.50
ctrl_f2	418	0.08	0.04	0.00	0.09	0.12	0.17	0.24
ctrl_f3	304	1.48	1.70	0.06	0.10	3.57	5.56	8.10
ctrl_f4	12	0.08	0.01	0.07	0.07	0.08	0.09	0.09
ctrl_f6	89	0.20	0.04	0.16	0.18	0.30	0.36	0.42
delete	67	0.00	0.00	0.00	0.00	0.00	0.00	0.00
down	7,546	0.00	0.01	0.00	0.00	0.01	0.02	0.12
end	4,526	0.66	3.01	0.00	0.14	1.59	8.67	42.38
enter	59,444	0.13	4.76	0.00	0.01	0.12	0.46	875.58
erase	717	0.00	0.01	0.00	0.00	0.01	0.01	0.07
f10	5,098	0.08	0.09	0.00	0.08	0.16	0.22	2.89
f2	2,528	0.03	0.02	0.00	0.01	0.06	0.06	0.22
f3	3,598	0.55	1.05	0.04	0.08	3.02	4.84	16.86
f4	7,338	0.19	0.28	0.01	0.07	0.67	1.43	3.07
f7	676	0.02	0.01	0.01	0.02	0.03	0.06	0.20
f9	1,417	0.02	0.02	0.00	0.03	0.05	0.10	0.27
other	9,572	0.01	0.11	0.00	0.00	0.02	0.11	8.00
page_down	371	0.07	0.04	0.00	0.09	0.11	0.13	0.18
page_up	1,352	0.05	0.02	0.02	0.06	0.07	0.10	0.38
shift_tab	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00
tab	1,322	0.01	0.01	0.00	0.00	0.04	0.05	0.12
tms	2,014	23.28	0.66	22.62	23.09	24.48	26.67	27.96
up	169	0.00	0.04	0.00	0.00	0.01	0.01	0.07
Unspecified	6,042	0.28	0.44	0.00	0.05	0.81	2.31	6.14
Overall	**116,545**	**0.55**	**4.59**	**0.00**	**0.02**	**0.68**	**23.08**	**875.58**

For reasons already noted, scripts were designed to be as independent as possible; each begins with a user login and terminates with a user logout. This is not how CEFMS, and most other DBMSs, are accessed. Some CEFMS activities were not included in the mix simply because there was not time to prepare scripts for them. Finally, many decisions concerning mix content and size were made because of limitations imposed by the RTE and operating environment. All of these issues must be addressed so that the tests carry as much credibility as possible.

Finally, the work reported herein is of a "proof-of-concept" nature. The methodology and the tools developed here should be thoroughly exercised by conducting a full-fledged scientific experiment. Such an experiment would require the use of statistical experimental design techniques. Specifically, several hardware and software factors, which could potentially impact performance, should be selected, and an analysis of variance approach should be used to test the effects of those factors. Such an experiment would very probably require dozens of BTs and significant amounts of manpower, machine time, and disk space.

References

Adams, J. C., Currie, W. S., and Gilmour, B. A. C. (1978). "The structure and uses of the Edinburgh Remote Terminal emulator," *Software-Practice and Experience* 8, 451-59.

Bailey, D. H., and Barton, J. T. (1985). "The NAS kernel benchmark program," NASA Technical Memorandum 86711, National Aeronautics and Space Administration, Ames Research Center, Moffett Field, California.

Buchholz, W. A. (1969). "A synthetic job for measuring system performance," IBM Systems Journal 8(4), 309-18.

Curnow, H. J., and Wichman, B. A. (1976). "A synthetic benchmark," *The Computer Journal* 19(1), 43-49.

Cybenko, G. (1990). "Supercomputer performance evaluation and the perfect club." *Proceedings of the 1990 International Conference on Supercomputing*. Association for Computing Machinery, New York, 254-66.

Dongarra, J. J., Martin, J. L., and Worlton, J. (1987). "Computer benchmarking: paths and pitfalls," *IEEE Spectrum* 24(7), 38-43.

Dougherty, D., and Robbins, A. (1997). *sed & awk*. O'Reilly & Associates, Sebastopol, California.

Federal Computer Performance Evaluation and Simulation Center. (1979). "Use and specifications of remote terminal emulation in ADP system acquisitions," FPR 1-4.11, General Services Administration, Automated Data and Telecommunications Services, Washington, DC.

Ferrari, D. (1972). "Workload characterization and selection in computer performance measurement," *Computer* 5(4), 18-24.

Grassl, C. M., and Schwarzmeier, J. L. (1990). "A new measure of supercomputer performance: results from the perfect benchmarks," *Cray Channels* 12(1), 14-16.

Gustafson, J., Rover, D., Elbert, S., and Carter, M. (1990). "SLALOM: the first scalable supercomputer benchmark," *Supercomputing Review*, 56-61.

Jones, R. (1975). "A survey of benchmarking: the state of the art." *Benchmarking: computer evalaution and measurement.* N. Benwell, ed., Hemisphere Publishing Corporation, Washington, DC, 15-23.

Kernighan, B. W., and Hamilton, P. A. (1973). "Synthetically generated performance test loads for operating systems." *1st Annual SIGME Symposium on Measurement and Evaluation.* Association for Computing Machinery, New York, 121-26.

McMahon, F. (1986). "The Livermore Fortran kernels: computer test of the numerical performance range," Technical Report UCRL-53745, Lawrence Livermore National Laboratory, Livermore, California.

Pure Software. (1996). *PurePerformix/TTY user's guide.* Pure Software, Sunnyvale, California.

Randell, B., and Russell, L. J. (1960). *ALGOL 60 implementation.* Academic Press, London.

Saavedra-Barrera, R. H. (1990). "The SPEC and perfect club benchmarks: promises and limitations." *Hot Chips Symposium 2.* Santa Clara, California.

Sreenivasan, K., and Kleinman, A. J. (1974). "On the construction of a representative synthetic workload," *Communications of the ACM* 17(3), 127-33.

Standard Performance Evaluation Corporation. (1997). Welcome to SPEC, http://www.specbench.org/.

Weicker, R. P. (1984). "Dhrystone: a synthetic systems programming benchmark," *Communications of the ACM* 27(10), 1013-30.

_____. (1988). "Dhrystone benchmark: rationale for version 2 and measurement rules," *SIGPLAN Notices* 23(8), 49-62.

Wood, D. C., and Forman, E. H. (1971). "Throughput measurement using a synthetic job stream." *1971 Fall Joint Computing Conference.* AFIPS Conference Proceedings, 39, AFIPS Press, Montvale, NJ, 51-56.

Appendix A
Software Tools Developed for This Project

This appendix lists many of the major software tools developed as part of this project. The first, *prccertf.sql* is an SQL script which counts the number of PR&C certifications performed during a specified time period. It is an example of numerous, but similar, SQL scripts written to obtain daily transaction counts. The remaining tools come in pairs. The first of a pair is a Bourne shell script which serves as a "wrapper" for the **awk** script which follows. All of the **awk** scripts call utility scripts to perform various housekeeping functions such as displaying error messages, manipulating linked lists, performing date conversion, and the like. These utility scripts are not listed here.

prccertf.sql

```
set pagesize 60
set echo off
set term off
set feedback off
set linesize 21
ttitle 'PR&C CERTIFICATIONS'
column trunc(a.cert_date) heading 'DATE' format a12
column count(b.prac_line_no) heading 'COUNT' format 9999999
spool prac_cert_man.lst
select trunc(a.cert_date), count(b.prac_line_no)
from  pr_amend a, pr_line_item b
where (b.moa_code <> 'I2'
      or b.travel_order_no is null
      or a.other_purchases_code <> 'CCRD')
and   trunc(a.cert_date) between '01-SEP-95' and '30-SEP-95'
and   a.prac_no = b.prac_no
group by trunc(a.cert_date)
/
spool off
exit
```

cpdir

```
#! /bin/sh
#-----------------------------------------------------------------
#
#    NAME
#         cpdir
#
#    SYNOPSIS
#         cpdir -f map_file [-ln]
#              [-ss source_suffix] [-ts target_suffix]
#              source_dir target_dir
#
#    DESCRIPTION
#         Copy files in source_dir to target_dir.
#
#    OPTIONS
#         -mf map_file   Columns one and two of map_file contain
#                        the old and new names of the data files.
#
#         -ln            Link instead of copy the files.
#
#         -ss source_suffix
#                        Append source_suffix to each source file
#                             name.
#
#         -ts target_suffix
#                        Append target_suffix to each target file
#                        name.
#
#         source_dir     Pathname of the directory containing the
#                        source files to be copied.
#
#         target_dir     Pathname of the directory which will
#                        contain the copies.
#
#    AUTHOR
#         William A. Ward, Jr., University of South Alabama.
#
#-----------------------------------------------------------------

CMD=`basename $0`
CMD_DIR=`dirname $0`
AWK1="$AWK"
AWK1="$AWK1 -f $CMD_DIR/prerr.awk"
AWK1="$AWK1 -f $CMD_DIR/cpdir.awk"
AWK1="$AWK1 X $*"
eval $AWK1
```

cpdir.awk

```
#! /bin/nawk -f
#-----------------------------------------------------------------
#
#    NAME
#         cpdir.awk
#
#    SYNOPSIS
#         [awk -f] cpdir X -f map_file [-ln] [-ss source_suffix]
#              [-ts target_suffix] source_dir target_dir
#
#    DESCRIPTION
#         Copy files in source_dir to target_dir.
```

```
#
#    OPTIONS
#         See cpdir for a description of the options.
#
#    SEE ALSO
#         cpdir
#
#    AUTHOR
#         William A. Ward, Jr., University of South Alabama.
#
#-----------------------------------------------------------------

#-----------------------------------------------------------------
#
#    FUNCTION
#         cpdir_getargs
#
#    PURPOSE
#         To get input arguments off of the command line.
#
#-----------------------------------------------------------------

function cpdir_getargs(         argc,i) {

     #       Set default values for arguments.

     CP_COMMAND    = "cp"
     SOURCE_SUFFIX = ""
     SOURCE_DIR    = ""
     TARGET_SUFFIX = ""
     TARGET_DIR    = ""
     VERBOSE       = 0

     #       Process the arguments in the array ARGV.

     argc = ARGC
     ARGC = 1
     for ( i=2; i<argc; i++ )
          if ( ARGV[i] == "-mf" )
               ARGV[ARGC++] = ARGV[++i]
          else if ( ARGV[i] == "-ln" )
               CP_COMMAND = "ln -s"
          else if ( ARGV[i] == "-ss" )
               SOURCE_SUFFIX = ARGV[++i]
          else if ( ARGV[i] == "-ts" )
               TARGET_SUFFIX = ARGV[++i]
          else if ( ARGV[i] == "-v" )
               VERBOSE = 1
          else if ( substr(ARGV[i],1,1) == "-" )
               prerr( "cpdir: Invalid argument " ARGV[i])
          else if ( SOURCE_DIR == "" )
               SOURCE_DIR = ARGV[i]
          else if ( TARGET_DIR == "" )
               TARGET_DIR = ARGV[i]
          else
               prerr( "cpdir: Two directory name arguments
               required" )

#    Check for errors in the input arguments.

     if ( ARGC == 1 )
          prerr( "cpdir: Required argument mf not supplied" )
     if ( SOURCE_DIR == "" )
          prerr( "cpdir: Two directory name arguments required" )
     if ( system( "test -d " SOURCE_DIR ) )
          prerr("cpdir: Source directory" SOURCE_DIR " does not
          exist" )
```

```
     if ( TARGET_DIR == "" )
          prerr( "cpdir: Two directory name arguments required" )
     if ( system( "test ! -f " TARGET_DIR ) )
          prerr( "cpdir: Target " TARGET_DIR "is not a directory"
          )
}

#-------------------------------------------------------------------
#
#    FUNCTION
#         cpdir_init
#
#    PURPOSE
#         To initialize variables and prepare for main loop.
#
#-------------------------------------------------------------------

function cpdir_init(    command) {

     #    If the target directory does not exist, create it.

     if ( system( "test -d " TARGET_DIR ) ) {
          command = "mkdir " TARGET_DIR
          if ( VERBOSE )
               print command
          system( command )
     }
}

#-------------------------------------------------------------------
#
#    ROUTINE
#         Main input loop
#
#    PURPOSE
#         Performed for every input line.
#
#-------------------------------------------------------------------

function cpdir_main(    command,source_file,target_file) {

     #    Strip out comments and skip blank lines.

     sub( /#.*/, "" )
     sub( /[          ]+$/, "" )
     if ( $0 != "" ) {

          #       Build the source and target file names.

          source_file = $1
          if ( $2 == "" )
               target_file = $1
          else
               target_file = $2
          source_file = SOURCE_DIR "/" source_file SOURCE_SUFFIX
          target_file = TARGET_DIR "/" target_file TARGET_SUFFIX

          #       Build the copy command and execute it.

          command = CP_COMMAND " " source_file " " target_file
          if ( VERBOSE )
               print command
          system( command )
     }
}
```

```
#-------------------------------------------------------------
#
#     ROUTINE
#          Main program.
#
#     PURPOSE
#          Top level procedure invocations.
#
#-------------------------------------------------------------

BEGIN {                         # Initialization.
     cpdir_getargs()            # Process command line arguments.
     cpdir_init()               # Initialize variables.
}
{                               # Performed once for each input line.
     cpdir_main()               # Copy file.
}
```

mt2mc

```
#! /bin/sh
#-------------------------------------------------------------
#
#     NAME
#          mt2mc (Mix Table To Mix Command file)
#
#     SYNOPSIS
#          mt2mc [-dtp time] [-dtr time] [-dts time] [-mode number]
#               [-nsu n|all][-th time][-tl time][-to time][-tr cps]
#               [file ...]
#
#     DESCRIPTION
#          Given a PurePerformix(tm)-compatible mix table, mt2mc
#          produces a PurePerformix-compatible mix command file. If
#          no input files are specified, mt2mc reads from standard
#          input.  Output is sent to standard output.
#
# OPTIONS
#     -dtp time      "Delta T Pause". If scripts are started and
#                    then suspended, then there is a pre-test pause
#                    of time seconds before they are resumed. The
#                    default value of time is 60.
#
#     -dtr time      "Delta T Resume".  If scripts are suspended,
#                    then after the pre-test pause (see -dtp), they
#                    are resumed time seconds apart.  Any other
#                    scripts started after the pause also use this
#                    time delay.  The default value of time is 1.
#
#     -dts time      "Delta T Start".  Scripts started before the
#                    pre-test pause (see -dtp) are separated by time
#                    seconds.  The default value of time is 1.
#
#     -mode number   Operating mode.  If number is 1, then a
#                    standard mix command file is produced.  If mode
#                    is 2, then a shell script to time the scripts
#                    is produced.
#
#     -nsu n|all     "Number of Suspended Users".  This specifes how
#                    many users to start in a suspended state prior
#                    to the actual start of the test. The default
#                    value is "all".
#
#     -th time       "Think high". Upper bound on randomly
#                    distributed tink times.  The default value is
```

```
#                   1.5 seconds.
#
#    -tl time       "Think low".  Lower bound on randomly
#                   distributed tink times.  The default value is
#                   0.5 seconds.
#
#    -to time       "Time out". Scripts will exit after time
#                   seconds.  The default value is 600 seconds.
#
#    -tr cps        "Type rate".  The default is 5 characters per
#                   second.
#
#    AUTHOR
#         William A. Ward, Jr., University of South Alabama.
#
#-------------------------------------------------------------------

CMD=`basename $0`
CMD_DIR=`dirname $0`
AWK1="$AWK"
AWK1="$AWK1 -f $CMD_DIR/prerr.awk"
AWK1="$AWK1 -f $CMD_DIR/randij.awk"
AWK1="$AWK1 -f $CMD_DIR/mt2mc.awk"
AWK1="$AWK1 X $*"
eval $AWK1
```

mt2mc.awk

```
#! /bin/awk -f
#-------------------------------------------------------------------
#
#    NAME
#         mt2mc.awk (Mix Table To Mix Comands)
#
#    SYNOPSIS
#         [awk -f] mt2mc.awk X [-dtp time] [-dtr time]
#              [-dts time] [-mode number] [-nsu n|all] [-th time]
#              [-tl time] [-to time] [-tr cps] [file ...]
#
#    DESCRIPTION
#         mt2mc.awk produces a PurePerformix(tm)-compatible mix
#         command file given an PurePerformix mix table.
#         mt2mc.awk is usually invoked from the shell script
#         "wrapper" mt2mc.
#
#    OPTIONS
#         See mt2mc for a description of the options.
#
#    SEE ALSO
#         mt2mc
#
#    AUTHOR
#         William A. Ward, Jr., University of South Alabama.
#
#-------------------------------------------------------------------

#-------------------------------------------------------------------
#
#    FUNCTION
#         mt2mc_getargs
#
#    PURPOSE
#         To get input arguments off of the command line.
#
```

```
#-----------------------------------------------------------------
function mt2mc_getargs( argc,i) {

     #       Set default values for arguments.

     DTP  = 60
     DTR  = 1
     DTS  = 1
     MODE = 1
     NSU  = "all"
     TH   = 1.5
     TL   = 0.5
     TO   = 600
     TR   = 5

     #       Process the arguments in the array ARGV.

     argc = ARGC
     ARGC = 1
     for ( i=2; i<argc; i++ )
          if ( ARGV[i] == "-dtp" )
               DTP = ARGV[++i]
          else if ( ARGV[i] == "-dtr" )
               DTR = ARGV[++i]
          else if ( ARGV[i] == "-dts" )
               DTS = ARGV[++i]
          else if ( ARGV[i] == "-mode" )
               MODE = ARGV[++i]
          else if ( ARGV[i] == "-nsu" )
               NSU = ARGV[++i]
          else if ( ARGV[i] == "-th" )
               TH = ARGV[++i]
          else if ( ARGV[i] == "-tl" )
               TL = ARGV[++i]
          else if ( ARGV[i] == "-to" )
               TO = ARGV[++i]
          else if ( ARGV[i] == "-tr" )
               TR = ARGV[++i]
          else if (substr(ARGV[i],1,1) == "-" && length(ARGV[i])
          >1)
               prerr( "mt2mc: Invalid argument " ARGV[i])
          else
               ARGV[ARGC++] = ARGV[i]

     #       Check for errors in the input arguments.

     if ( DTP < 0 )
          prerr( "mt2mc: Argument dtp is invalid" )
     if ( DTR < 0 )
          prerr( "mt2mc: Argument dtr is invalid" )
     if ( DTS < 0 )
          prerr( "mt2mc: Argument dts is invalid" )
     if ( MODE < 1 )
          prerr( "mt2mc: Argument mode is invalid" )
     if ( NSU < 0 )
          prerr( "mt2mc: Argument nsu is invalid" )
}

#-----------------------------------------------------------------
#
#    FUNCTION
#         mt2mc_init
#
#    PURPOSE
#         To check for errors in the input arguments.
#
#-----------------------------------------------------------------
```

```
function mt2mc_init() {
}

#-----------------------------------------------------------------
#
#    FUNCTION
#         mt2mc_mode1_begin
#
#    PURPOSE
#         Executed once inside BEGIN when mode = 1.
#
#-----------------------------------------------------------------
function mt2mc_mode1_begin() {
}

#-----------------------------------------------------------------
#
#    FUNCTION
#         mt2mc_mode1
#
#    PURPOSE
#         To process input lines in mode 1.
#
#-----------------------------------------------------------------

function mt2mc_mode1() {
     if ( $1 != "+" ) {
          n_rte_users = n_rte_users + 1
          rte_user[n_rte_users] = substr( $1, 1, length($1)-1 )
     }
}

#-----------------------------------------------------------------
#
#    FUNCTION
#         mt2mc_mode1_end
#
#    PURPOSE
#         Executed once inside END when mode = 1.
#
#-----------------------------------------------------------------

function mt2mc_mode1_end() {
     if ( NSU == "all" || NSU > n_rte_users )
          NSU = n_rte_users

     #       Print mix command header.

     print "! echo Mix execution begins"
     print "use " FILENAME
     print "! gv_reset"
     print "! gv_init SUSPEND int 0"
     print "! gv_init TH double " TH
     print "! gv_init TL double " TL
     print "! gv_init TO int " TO
     print "! gv_init TR int " TR

     #     Print these commands if some users are to be suspended.

     if ( NSU > 0 ) {

          #       Start users who will be suspended.

          print "! gv_write SUSPEND 1"

          if ( NSU == n_rte_users ) {
```

```
                print "set tstart " DTS
                print "start all"
                print "! echo All users started"
           } else {
                for ( i_rte_user=1; i_rte_user<=NSU; i_rte_user++){
                     print "pause " DTS
                     print "start " rte_user[i_rte_user]
                }
                print "! echo " NSU " users started"
           }

           #        Pre-test pause and reset suspend flag.

           print "pause " DTP
           print "! gv_write SUSPEND 0"
           print "! echo Benchmark test begins"

           #       Resume all suspended users.

           if ( NSU == n_rte_users ) {
                print "set tresume " DTR
                print "resume all"
                print "! echo All users resumed"
           } else {
                for (i_rte_user=1; i_rte_user<=NSU; i_rte_user++) {
                     print "pause " DTR
                     print "resume " rte_user[i_rte_user]
                }
                print "! echo " NSU " users resumed"
           }
      }

      #       If there are any other users, start them.

      if ( n_rte_users > NSU ) {
           if ( NSU == 0 ) {
                print "set tstart " DTR
                print "start all"
                print "! echo All users started"
           } else {
                for ( i_rte_user=NSU+1; i_rte_user<=n_rte_users; \
                     i_rte_user++) {
                     print "pause " DTR
                     print "start " rte_user[i_rte_user]
                }
                print "! echo All remaining users started"
           }
      }

      #       Wait until all users complete and then quit.

      print "wait"
      print "! echo Benchmark test ends"
      print "quit"
}

#-----------------------------------------------------------------
#
#    FUNCTION
#         mt2mc_mode2_begin
#
#    PURPOSE
#         Executed once inside BEGIN when mode = 2.
#
#-----------------------------------------------------------------

function mt2mc_mode2_begin() {
```

```
     time_command = "timecmd "
     print "#! /bin/sh"
}

#-------------------------------------------------------------------
#
#    FUNCTION
#         mt2mc_mode2
#
#    PURPOSE
#         To process input lines in mode 2.
#
#-------------------------------------------------------------------

function mt2mc_mode2() {
     #       Delete the user name or "+ sleep <n>".

     if ( $1 != "+" )
          sub( /^[^ ]*, */, "" )
     else {
          sub( /^\+ +/, "" )
          sub( /^sleep +[0-9]+ +/, "" )
     }

     #       Replace with the timing command.

     sub( /^/, time_command )
     print $0
}

#-------------------------------------------------------------------
#
#    FUNCTION
#         mt2mc_mode2_end
#
#    PURPOSE
#         Executed once inside END when mode = 2.
#
#-------------------------------------------------------------------

function mt2mc_mode2_end() {
}

#-------------------------------------------------------------------
#
#    ROUTINE
#         BEGIN
#
#    PURPOSE
#         Top level procedure invocations.
#
#-------------------------------------------------------------------

BEGIN {                    # Initialization.
     mt2mc_getargs()       # Process command line arguments.
     mt2mc_init()          # Initialize variables.

     if ( MODE == 1 )
          mt2mc_mode1_begin()
     else
          mt2mc_mode2_begin()
}
{                          # Performed once for every input line.
     if ( MODE == 1 )
          mt2mc_mode1()
     else
```

```
          mt2mc_mode2()
}
END {                          # Wrap up at end.
     if ( error )
          exit
     else if ( MODE == 1 )
          mt2mc_mode1_end()
     else
          mt2mc_mode2_end()
}
```

mt2mt0

```
#! /bin/sh
#-----------------------------------------------------------------
#
#    NAME
#         mt2mt0 (Mix Table to Mix Table Zero)
#
#    SYNOPSIS
#         mt2mt0 -suf file -uwf file [file ...]
#
#    DESCRIPTION
#         Given an PurePerformix(r)-compatible mix table for a
#         benchmark test, mt2mt0 produces a second mix table of
#         scripts which must be executed prior to the test in
#         order for the test scripts to execute correctly.
#
#    OPTIONS
#         -suf file   Name of the script-user file. The first
#                     field of sufile is a script name.  Following
#                     fields are user names which can execute that
#                     script.  Fields are separated by blanks or
#                     tabs.  Blank lines and comments (beginning
#                     with "#" are ignored.
#
#         -uwf file   Name of the user-work item file. The first
#                     field field of uwfile is a username.
#                     Following fields are work item codes owned
#                     by that user. Fields are separated by blanks
#                     or tabs.  Blank lines and comments
#                     (beginning with "#" are ignored.
#
#    AUTHOR
#         William A. Ward, Jr., University of South Alabama.
#

#    BUGS
#         This is not a general-purpose script.  It will only
#         handle scripts specific to the CEFMS benchmark test as
#         developed at the U. S. Army Engineer Waterways
#         Experiment Station (WES) in Summer 1996.
#
#-----------------------------------------------------------------

CMD=`basename $0`
CMD_DIR=`dirname $0`
AWK1="$AWK"
AWK1="$AWK1 -f $CMD_DIR/date.awk"
AWK1="$AWK1 -f $CMD_DIR/ltab.awk"
AWK1="$AWK1 -f $CMD_DIR/prerr.awk"
AWK1="$AWK1 -f $CMD_DIR/prmt.awk"
AWK1="$AWK1 -f $CMD_DIR/randij.awk"
AWK1="$AWK1 -f $CMD_DIR/mt2mt0.awk"
```

```
AWK1="$AWK1 X $*"
eval $AWK1
```

mt2mt0.awk

```
#! /bin/awk -f
#-----------------------------------------------------------------
#
#    NAME
#         mt2mt0.awk (Mix Table To Mix Table Zero)
#
#    SYNOPSIS
#         [awk -f] mt2mt0.awk X -suf file -uwf file [file ...]
#
#    DESCRIPTION
#         mt2mt0.awk produces a PurePerforix(tm)-compatible
#         preparatory mix table given a mix table to be used in a
#         benchmark test. mt2mt0.awk is usually invoked from the
#         shell script "wrapper" mt2mt0.
#
#    OPTIONS
#         See mt2mt0 for a description of the options.
#
#    SEE ALSO
#         mt2mt0
#
#    AUTHOR
#         William A. Ward, Jr.
#
#    BUGS
#         This is not a general-purpose script.  It will only
#         handle scripts specific to the CEFMS benchmark test as
#         developed at the U. S. Army Engineer Waterways
#         Experiment Station (WES) in Summer 1996.
#
#-----------------------------------------------------------------

#-----------------------------------------------------------------
#
#    FUNCTION
#         mt2mt0_getargs
#
#    PURPOSE
#         To get input arguments off of the command line.
#
#-----------------------------------------------------------------
function mt2mt0_getargs( argc,i) {

     #       Process the arguments in the array ARGV.

     argc = ARGC
     ARGC = 1
     for ( i=2; i<argc; i++ )
          if ( ARGV[i] == "-suf" )
               SUF = ARGV[++i]
          else if ( ARGV[i] == "-uwf" )
               UWF = ARGV[++i]
          else if ( substr(ARGV[i],1,1) == "-" && length(ARGV[i])
          > 1 )
               prerr( "mt2mt0: Invalid argument " ARGV[i] )
          else
               ARGV[ARGC++] = ARGV[i]

     #       Check for errors in the input arguments.
```

```
     if ( SUF == "" )
          prerr( "mt2mt0: Required argument suf not supplied" )
     if ( UWF == "" )
          prerr( "mt2mt0: Required argument uwf not supplied" )
}

#-------------------------------------------------------------------
#
#    FUNCTION
#         mt2mt0_date
#
#    PURPOSE
#         To construct date strings.
#
#-------------------------------------------------------------------

function mt2mt0_date(n  ,yyyymmdd) {

     yyyymmdd = date_n2yyyymmdd( n )
     return  substr(yyyymmdd,7,2) \
       "-" tolower(substr(date_n2moty(substr(yyyymmdd,5,2)),1,3))\
       "-" substr(yyyymmdd,3,2)
}

#-------------------------------------------------------------------
#
#    FUNCTION
#         mt2mt0_init
#
#    PURPOSE
#         To perform initialization of variables after argument
#         processing.
#
#-------------------------------------------------------------------

function mt2mt0_init(   n,yyyymmdd) {

     date_n2moty_init()
     date_n2yyyymmdd_init()
     date_yyyymmdd2n_init()
     "date +%Y%m%d" | getline yyyymmdd  # yyyymmdd is current date
     n = date_yyyymmdd2n( yyyymmdd )  # n is number of current day
     INPF[1,"torcreat"]=mt2mt0_date(n+365) # travel order depart
     date
     INPF[2,"torcreat"]=mt2mt0_date(n+366) #travel order return
     date
     INPF[1,"torcrevo"]=mt2mt0_date(n+1) #travel voucher depart
     date
     INPF[2,"torcrevo"]=mt2mt0_date(n+2) #travel voucher return
     date
}

#-------------------------------------------------------------------
#
#    FUNCTION
#         mt2mt0_prmt
#
#    PURPOSE
#         To print one line of the preparatory mix table.
#
#-------------------------------------------------------------------

function mt2mt0_prmt(script,port,user,inpf,outf,udif) {

     if ( user == "" ) {
          do
               user = ltab_getval( SU_LTAB, script, "RANDOM" )
```

```
          while ( index( udif, user ) )
     }
     prmt_line( script, port, user, "p", inpf, outf)
}

#-----------------------------------------------------------------
#
#    FUNCTION
#         mt2mt0_mkinpf
#
#    PURPOSE
#         To make the input file for a script.
#
#-----------------------------------------------------------------

function mt2mt0_mkinpf(inpf,field1,field2,field3,field4) {

     if ( inpf != "" ) {
          inpf = inpf ".d"
          print field1 > inpf
          if ( field2 != "" ) {
               print field2 >> inpf
               if ( field3 != "" ) {
                    print field3 >> inpf
                    if ( field4 != "" )
                         print field4 >> inpf
               }
          }
          close( inpf )
     }
}

#-----------------------------------------------------------------
#
#    FUNCTION
#         mt2mt0_cor
#
#    PURPOSE
#         To process cor type scripts.
#
#-----------------------------------------------------------------

function mt2mt0_cor(script,port,user,inpf \
     ,approver,corcreat_inpf,work_item) {

     if ( script == "corcreat" ) {
          approver = ltab_getval( SU_LTAB, "corapprv", "RANDOM" )
          work_item = ltab_getval( UW_LTAB, approver, "RANDOM" )
          mt2mt0_mkinpf( inpf, work_item )

     } else if ( script == "corapprv" ) {
          mt2mt0_prmt( "corcreat", port, "", "", inpf )
          corcreat_inpf = prmt_query( "inpf" )
          approver  = user
          work_item = ltab_getval( UW_LTAB, approver, "RANDOM" )
          mt2mt0_mkinpf( corcreat_inpf, work_item )

     } else if ( script == "corcertf" ) {
          mt2mt0_prmt( "corcreat", port, "", "", inpf )
          corcreat_inpf = prmt_query( "inpf" )
          mt2mt0_prmt( "corapprv", port, "", inpf )
          approver  = prmt_query( "user" )
          work_item = ltab_getval( UW_LTAB, approver, "RANDOM" )
          mt2mt0_mkinpf( corcreat_inpf, work_item )

     } else if ( script == "coraccpt" ) {
```

```
          mt2mt0_prmt( "corcreat", port, "", "", inpf )
          corcreat_inpf = prmt_query( "inpf" )
          mt2mt0_prmt( "corapprv", port, "", inpf )
          approver  = prmt_query( "user" )
          work_item = ltab_getval( UW_LTAB, approver, "RANDOM" )
          mt2mt0_mkinpf( corcreat_inpf, work_item )
          mt2mt0_prmt( "corcertf", port, "", inpf )
     }
}

#-----------------------------------------------------------------
#
#    FUNCTION
#         mt2mt0_cvo
#
#    PURPOSE
#         To process cvo type scripts.
#
#-----------------------------------------------------------------

function mt2mt0_cvo(script,port,user,inpf \
     ,approver,cvocreat_inpf,work_item) {

     if ( script == "cvocreat" ) {
          approver = ltab_getval( SU_LTAB, "cvocertf", "RANDOM" )
          work_item = ltab_getval( UW_LTAB, approver, "RANDOM" )
          mt2mt0_mkinpf( inpf, work_item )

     } else if ( script == "cvocertf" ) {
          mt2mt0_prmt( "cvocreat", port, "", "", inpf )
          cvocreat_inpf = prmt_query( "inpf" )
          approver  = user
          work_item = ltab_getval( UW_LTAB, approver, "RANDOM" )
          mt2mt0_mkinpf( cvocreat_inpf, work_item )
     }
}

#-----------------------------------------------------------------
#
#    FUNCTION
#         mt2mt0_lab
#
#    PURPOSE
#         To process lab type scripts.
#
#-----------------------------------------------------------------

function mt2mt0_lab(script,port,user,inpf \
     ,approver,work_item) {

     if ( script == "labcreat" ) {
          work_item = ltab_getval( UW_LTAB, user, "RANDOM" )
          mt2mt0_mkinpf( inpf, work_item )

     } else if ( script == "labctran" ) {
          # do nothing
     }
}

#-----------------------------------------------------------------
#
#    FUNCTION
#         mt2mt0_prc
#
#    PURPOSE
#         To process prc type scripts.
#
```

```
#-----------------------------------------------------------------

function mt2mt0_prc(script,port,user,inpf \
     ,approver,prccreat_inpf,work_item) {

     if ( script == "prccreat" ) {
          approver = ltab_getval( SU_LTAB, "prcapprv", "RANDOM" )
          work_item = ltab_getval( UW_LTAB, approver, "RANDOM" )
          mt2mt0_mkinpf( inpf, work_item )

     } else if ( script == "prcapprv" ) {
          mt2mt0_prmt( "prccreat", port, "", "", inpf )
          prccreat_inpf = prmt_query( "inpf" )
          approver  = user
          work_item = ltab_getval( UW_LTAB, approver, "RANDOM" )
          mt2mt0_mkinpf( prccreat_inpf, work_item )

     } else if ( script == "prccertf" ) {
          mt2mt0_prmt( "prccreat", port, "", "", inpf )
          prccreat_inpf = prmt_query( "inpf" )
          mt2mt0_prmt( "prcapprv", port, "", inpf )
          approver  = prmt_query( "user" )
          work_item = ltab_getval( UW_LTAB, approver, "RANDOM" )
          mt2mt0_mkinpf( prccreat_inpf, work_item )

     } else if ( script == "prccreob" ) {
          mt2mt0_prmt( "prccreat", port, "", "", inpf )
          prccreat_inpf = prmt_query( "inpf" )
          mt2mt0_prmt( "prcapprv", port, "", inpf )
          approver  = prmt_query( "user" )
          work_item = ltab_getval( UW_LTAB, approver, "RANDOM" )
          mt2mt0_mkinpf( prccreat_inpf, work_item )
          mt2mt0_prmt( "prccertf", port, "", inpf )

     } else if ( script == "prcappob" ) {
          mt2mt0_prmt( "prccreat", port, "", "", inpf )
          prccreat_inpf = prmt_query( "inpf" )
          mt2mt0_prmt( "prcapprv", port, "", inpf )
          approver  = prmt_query( "user" )
          work_item = ltab_getval( UW_LTAB, approver, "RANDOM" )
          mt2mt0_mkinpf( prccreat_inpf, work_item )
          mt2mt0_prmt( "prccertf", port, "", inpf )
          mt2mt0_prmt( "prccreob", port, "", inpf )

     } else if ( script == "prccrerr" || script == "prccrein" ) {
          mt2mt0_prmt( "prccreat", port, "", "", inpf )
          prccreat_inpf = prmt_query( "inpf" )
          mt2mt0_prmt( "prcapprv", port, "", inpf )
          approver  = prmt_query( "user" )
          work_item = ltab_getval( UW_LTAB, approver, "RANDOM" )
          mt2mt0_mkinpf( prccreat_inpf, work_item )
          mt2mt0_prmt( "prccertf", port, "", inpf )
          mt2mt0_prmt( "prccreob", port, "", inpf )
          mt2mt0_prmt( "prcappob", port, "", inpf )
     }
}

#-----------------------------------------------------------------
#
#    FUNCTION
#         mt2mt0_rep
#
#    PURPOSE
#         To process rep type scripts.
#
#-----------------------------------------------------------------
```

```
function mt2mt0_rep(script,port,user,inpf \
     ) {

     if ( script == "repa3953" || \
          script == "repcertl" || \
          script == "repcolds" || \
          script == "repmscdb" || \
          script == "reptmatt" || \
          script == "repvstat" ) {
               # do nothing

     } else if ( script == "repa4445" ) {
          mt2mt0_mkinpf( inpf, 50 )
     }
}

#-----------------------------------------------------------------
#
#    FUNCTION
#         mt2mt0_tor
#
#    PURPOSE
#         To process tor type scripts.
#
#-----------------------------------------------------------------

function mt2mt0_tor(script,port,user,inpf \
     ,approver,torcreat_inpf,udif,work_item) {

     if ( script == "torcreat" ) {
          approver = ltab_getval( SU_LTAB, "torapprv", "RANDOM" )
          work_item = ltab_getval( UW_LTAB, approver, "RANDOM" )
          mt2mt0_mkinpf( inpf, work_item, \
               INPF[1,"torcreat"], INPF[2,"torcreat"] )

     } else if ( script == "torreque" ) {
          mt2mt0_prmt( "torcreat", port, "", "", inpf )
          torcreat_inpf = prmt_query( "inpf" )
          approver  = ltab_getval( SU_LTAB, "torapprv", "RANDOM" )
          work_item = ltab_getval( UW_LTAB, approver, "RANDOM" )
          mt2mt0_mkinpf( torcreat_inpf, work_item, \
               INPF[1,"torcreat"], INPF[2,"torcreat"] )

     } else if ( script == "torapprv" ) {
          mt2mt0_prmt( "torcreat", port, "", "", inpf, user )
          torcreat_inpf = prmt_query( "inpf" )
          mt2mt0_prmt( "torreque", port, "", inpf, "", user )
          approver  = user
          work_item = ltab_getval( UW_LTAB, approver, "RANDOM" )
          mt2mt0_mkinpf( torcreat_inpf, work_item, \
               INPF[1,"torcreat"], INPF[2,"torcreat"] )

     } else if ( script == "torauthn" ) {
          mt2mt0_prmt( "torcreat", port, "", "", inpf )
          torcreat_inpf = prmt_query( "inpf" )
          udif = prmt_query( "user" )
          mt2mt0_prmt( "torreque", port, "", inpf )
          udif = udif " " prmt_query( "user" )
          mt2mt0_prmt( "torapprv", port, "", inpf, "", udif )
          approver  = prmt_query( "user" )
          work_item = ltab_getval( UW_LTAB, approver, "RANDOM" )
          mt2mt0_mkinpf( torcreat_inpf, work_item, \
               INPF[1,"torcreat"], INPF[2,"torcreat"] )

     } else if ( script == "torcrevo" ) {
          mt2mt0_prmt( "torcreat", port, user, "", inpf )
          torcreat_inpf = prmt_query( "inpf" )
```

```
          mt2mt0_prmt( "torreque", port, "", inpf )
          udif = user " " prmt_query( "user" )
          mt2mt0_prmt( "torapprv", port, "", inpf, "", udif )
          approver  = prmt_query( "user" )
          work_item = ltab_getval( UW_LTAB, approver, "RANDOM" )
          mt2mt0_mkinpf( torcreat_inpf, work_item, \
               INPF[1,"torcrevo"], INPF[2,"torcrevo"] )
          mt2mt0_prmt( "torauthn", port, "", inpf )

     } else if ( script == "torappvo" ) {
          mt2mt0_prmt( "torcreat", port, "", "", inpf, user )
          torcreat_inpf = prmt_query( "inpf" )
          creator = prmt_query( "user" )
          udif = creator " " user
          mt2mt0_prmt( "torreque", port, "", inpf, "", udif )
          mt2mt0_prmt( "torapprv", port, user, inpf )
          work_item = ltab_getval( UW_LTAB, user, "RANDOM" )
          mt2mt0_mkinpf( torcreat_inpf, work_item, \
               INPF[1,"torcrevo"], INPF[2,"torcrevo"] )
          mt2mt0_prmt( "torauthn", port, "", inpf )
          mt2mt0_prmt( "torcrevo", port, creator, inpf )
     }
}

#-----------------------------------------------------------------
#
#    FUNCTION
#         mt2mt0_trn
#
#    PURPOSE
#         To process trn type scripts.
#
#-----------------------------------------------------------------

function mt2mt0_trn(script,port,user,inpf \
     ,approver,trncreat_inpf,work_item) {

     if ( script == "trncreat" ) {
          approver = ltab_getval( SU_LTAB, "trnapprv", "RANDOM" )
          work_item = ltab_getval( UW_LTAB, approver, "RANDOM" )
          mt2mt0_mkinpf( inpf, work_item )

     } else if ( script == "trnapprv" ) {
          mt2mt0_prmt( "trncreat", port, "", "", inpf )
          trncreat_inpf = prmt_query( "inpf" )
          approver  = user
          work_item = ltab_getval( UW_LTAB, approver, "RANDOM" )
          mt2mt0_mkinpf( trncreat_inpf, work_item )
     }
}

#-----------------------------------------------------------------
#
#    FUNCTION
#         mt2mt0_vis
#
#    PURPOSE
#         To process vis type scripts.
#
#-----------------------------------------------------------------
function mt2mt0_vis(script,port,user,inpf \
     ,approver,viscreat_inpf,work_item) {

     if ( script == "viscreat" ) {
          approver = ltab_getval( SU_LTAB, "visapprv", "RANDOM" )
          work_item = ltab_getval( UW_LTAB, approver, "RANDOM" )
```

```
          mt2mt0_mkinpf( inpf, work_item )

     } else if ( script == "visapprv" ) {
          mt2mt0_prmt( "viscreat", port, "", "", inpf )
          viscreat_inpf = prmt_query( "inpf" )
          approver  = user
          work_item = ltab_getval( UW_LTAB, approver, "RANDOM" )
          mt2mt0_mkinpf( viscreat_inpf, work_item )
     }
}

#-------------------------------------------------------------------
#
#    ROUTINE
#         BEGIN
#
#    PURPOSE
#         Executed before any input lines are read.
#
#-------------------------------------------------------------------

BEGIN {
     mt2mt0_getargs()            # Process command-line arguments.
     mt2mt0_init()               # Initailize variables.
     ltab_build( SU_LTAB, SUF)   # Build the script-user list.
     ltab_build( UW_LTAB, UWF)   # Build the user-work item list.
}

#-------------------------------------------------------------------
#
#    ROUTINE
#         Main input loop
#
#    PURPOSE
#         Performed for every input line.
#
#-------------------------------------------------------------------

{
     #    Load the fields from the mix table entry into variables;
     #    the input file is assumed to be the last field.

     script = $(NF-4)
     port   = $(NF-3)
     logf   = $(NF-2)
     user   = $(NF-1)
     inpf   = $(NF)

     #    If the first field is not a continuation line,
     #    reset so that the next output line starts a new user.

     if ( $1 != "+" )
          prmt_reset()

     #   Array mt2mt0_sc is used by mt2mt0_mmkinpf (called by END)
     #   to create any remaining input files.

     mt2mt0_sc[script] = mt2mt0_sc[script] + 1

     #   Use the first 3 chars of the script name to determine
     #   what type of processing should be done.

     type = substr(script,1,3)
     if ( type == "cor" )
          mt2mt0_cor( script, port, user, inpf )
     else if ( type == "cvo" )
          mt2mt0_cvo( script, port, user, inpf )
```

```
     else if ( type == "lab" )
          mt2mt0_lab( script, port, user, inpf )
     else if ( type == "prc" )
          mt2mt0_prc( script, port, user, inpf )
     else if ( type == "rep" )
          mt2mt0_rep( script, port, user, inpf )
     else if ( type == "tor" )
          mt2mt0_tor( script, port, user, inpf )
     else if ( type == "trn" )
          mt2mt0_trn( script, port, user, inpf )
     else if ( type == "tvo" )
          mt2mt0_tvo( script, port, user, inpf )
     else if ( type == "vis" )
          mt2mt0_vis( script, port, user, inpf )
}

#-------------------------------------------------------------------
#
#    ROUTINE
#         END
#
#    PURPOSE
#         To wrap-up after all input lines have been read.
#
#-------------------------------------------------------------------

END {
     if ( error )
          exit
}
```

pa2pf

```
#! /bin/sh
#-------------------------------------------------------------------
#
#    NAME
#         pa2pf (Process Accounting To Peaking Factor)
#
#    SYNOPSIS
#         pa2pf [-a h[:m[:s]]] [-b h[:m[:s]]] [-d h[:m[:s]]]
#              [-w h[:m[:s]]] [file ...]
#
#    DESCRIPTION
#         Process a pacct file to produce a peaking factor.
#
#    OPTIONS
#         -a h[:m[:s]]  Start time in hours (:minutes (:seconds));
#                       input data prior to this is ignored.
#                       The default is 00:00:00.
#
#         -b h[:m[:s]]  Stop time in hours (:minutes (:seconds));
#                       input data after this is ignored.
#                       The default is 24:00:00.
#
#         -d h[:m[:s]]  Delta time in hours (:minutes (:seconds));
#                       input data is grouped into chunks of this
#                       size.  The default is 00:01:00.
#
#         -w h[:m[:s]]  Time window in hours (:minutes
#                       (:seconds)); candidate peaking factors for
#                       each possible windows of this size are
#                       calculated to determine the maximum (i.e.,
#                       the windows are (a,a+w),(a+d,a+d+w),
```

```
#                       (a+2d,a+2d+w), ..., (b-w,b)). The default
#                       is 01:00:00.
#
#    OPERANDS
#         file ... Path name of one or more input files.  If no
#                  files are specified, the standard input will be
#                  read.
#
#    SEE ALSO
#         acctcom, wt2pf
#
#    AUTHOR
#         William A. Ward, Jr., University of South Alabama.
#
#---------------------------------------------------------------

CMD=`basename $0`
CMD_DIR=`dirname $0`
AWK1="$AWK"
AWK1="$AWK1 -f $CMD_DIR/prerr.awk"
AWK1="$AWK1 -f $CMD_DIR/date.awk"
AWK1="$AWK1 -f $CMD_DIR/pa2pf.awk"
AWK1="$AWK1 X $*"
eval $AWK1
```

pa2pf.awk

```
#! /bin/nawk -f
#---------------------------------------------------------------
#
#    NAME
#         pa2pf.awk
#
#    SYNOPSIS
#         [awk -f] pa2pf.awk X -a h[:m]] [-b h[:m]] [-d h[:m]]
#              [-w h[:m]] [file ...]
#
#    DESCRIPTION
#         pa2pf.awk processes a pacct file to produce a peaking
#         factor. pa2pf.awk is usually invoked from the shell
#         script "wrapper" pa2pf.
#
#    OPTIONS
#         See pa2pf for a description of the options.
#
#    SEE ALSO
#         acctcom, pa2pf, wt2pf
#
#    AUTHOR
#         William A. Ward, Jr., University of South Alabama.
#
#---------------------------------------------------------------

#---------------------------------------------------------------
#
#    FUNCTION
#         pa2pf_getargs
#
#    PURPOSE
#         To get input arguments off of the command line.
#
#---------------------------------------------------------------

function pa2pf_getargs( argc,i) {
```

```
        #       Set default values for arguments.

        A  = "00:00:00"
        B  = "24:00:00"
        D  = "00:01:00"
        W  = "01:00:00"

        #       Process the arguments in the array ARGV.

        argc = ARGC
        ARGC = 1
        for ( i=2; i<argc; i++ )
             if ( ARGV[i] == "-a" )
                  A = ARGV[++i]
             else if ( ARGV[i] == "-b" )
                  B = ARGV[++i]
             else if ( ARGV[i] == "-d" )
                  D = ARGV[++i]
             else if ( ARGV[i] == "-w" )
                  W = ARGV[++i]
             else if ( substr(ARGV[i],1,1) == "-" && length(ARGV[i])
             > 1 )
                  prerr( "pa2pf: Invalid argument " ARGV[i])
             else
                  ARGV[ARGC++] = ARGV[i]

        #       Argument postprocessing

        TA = date_hms2s(A)
        TB = date_hms2s(B)
        TD = date_hms2s(D)
        TW = date_hms2s(W)

        #       Check for errors in the input arguments.

        if ( TA >= TB )
             prerr( "pa2pf: Start time >= stop time" )
        if ( TW >= TB-TA )
             prerr( "pa2pf: Window size > interval" )
}

#-----------------------------------------------------------------
#
#    FUNCTION
#         pa2pf_init
#
#    PURPOSE
#         To check for errors in the input arguments.
#
#-----------------------------------------------------------------

function pa2pf_init(    i) {
        getline
        if ( NF == 0)
             getline
        if ( $1 == "ACCOUNTING" ) {
             getline
             getline
             getline
        }
        pa2pf_main()
}

#-----------------------------------------------------------------
#
#    FUNCTION
```

```
#         pa2pf_main
#
#    PURPOSE
#         Executed for every input line.
#
#-----------------------------------------------------------------

function pa2pf_main(    ) {

     # Fields 4 and 5 contain the start and stop time of the
     process

     pa = date_hms2s( $4 )
     pb = date_hms2s( $5 )
     if ( pa < PA_MIN )
          PA_MIN = pa
     if ( pb > PB_MIN )
          PB_MAX = pb

     #    Skip if out of range

     if ( pb < TA || TB < pa ) {
          #       Do nothing
     } else {
          ia = int( pa / TD )
          ib = int( pb / TD )

          #       Did the process start and stop
          #       in the same time interval?

          if ( ia == ib ) {
               CPU_SEC[ia] = CPU_SEC[ia] + $7
               ta = 0
               tm = $7
               tb = 0

          #       The process started and ended
          #       in different time intervals

          } else {
               unit = $7 / (pb - pa)
               CPU_SEC[ia] = CPU_SEC[ia] + ((ia+1)*TD - pa)*unit
               for ( i=ia+1; i<ib; i++ )
                    CPU_SEC[i] = CPU_SEC[i] + TD*unit
               CPU_SEC[ib] = CPU_SEC[ib] + (pb - ib*TD)*unit
               ta = ((ia+1)*TD - pa)*unit
               tm = (ib - ia - 1)*TD*unit
               tb = (pb - ib*TD)*unit
          }
          ts = ta + tm + tb
          #printf "%7.2f %7.2f %7.2f %7.2f %7.2f\n", ta, tm, tb,
          ts, $7
     }
}

#-----------------------------------------------------------------
#
#    FUNCTION
#         pa2pf_end
#
#    PURPOSE
#         Executed at end of program.
#
#-----------------------------------------------------------------

function pa2pf_end(     i,n) {
     ia = int( TA / TD )
```

```
        ib = int( TB / TD )
        iw = int( TW / TD )
        for ( i=ia; i<=ia+iw-1; i++ )
                area = area + CPU_SEC[i]
        wmax = area
        wcur = area
        for ( i=ia+iw; i<=ib; i++ )
                area = area + CPU_SEC[i]
        for ( i=ia; i<=ib-iw; i++) {
                wcur = wcur - CPU_SEC[i] + CPU_SEC[i+iw]
                if ( wcur > wmax )
                        wmax = wcur
        }
        wavg = area / (TB - TA)
        wmax = wmax / TW
        pf = wmax / wavg
        printf "%9.3f %9.3f %8.3f\n", wmax, wavg, pf
}

#-----------------------------------------------------------------
#
#       ROUTINE
#               Main program.
#
#       PURPOSE
#               Top level procedure invocations.
#
#-----------------------------------------------------------------

BEGIN {                         # Initialization.
        pa2pf_getargs()         # Process command-line arguments.
        pa2pf_init()            # Initialize variables.
}
{                               # Performed once for each input line.
        pa2pf_main()
}
END {                           # Wrap up at end.
        if ( error )
                exit
        pa2pf_end()
}
```

sc2mt

```
#! /bin/sh
#-----------------------------------------------------------------
#
#       NAME
#               sc2mt (Script Count To Mix Table)
#
#       SYNOPSIS
#               sc2mt [-mode number] [-nspu number] [-nu number] [-suf
                file]
#                       [-sut host] [file ...]
#
#       DESCRIPTION
#               Given a list of script names and counts, sc2mt produces
#               a PurePerformix(tm)-compatible mix table.
#
#               Each line of input contains a script name and a count
#               specifying how many times each script is to executed
#               in the RTE-driven test.
#
#       OPTIONS
```

```
#          -mode number  If number = 1, the input script counts are
#                        used to determine how many times each
#                        ecript is executed.  If number = 2, the
#                        input script counts are ignored and each
#                        script is executed once by each valid
#                        username in the script-user file.
#
#          -nspu number  Number of scripts per (emulated) user.
#                        Let ns and nu denote the number of scripts
#                        and number of users respectively; then nu
#                        = ns / nspu.  If both-nspu and -nu are
#                        specified, -nu takes precedence. The
#                        default value of nspu is 1.
#
#          -nu number    Number of (emulated) users.  Let ns and
#                        nspu denote the number of scripts and
#                        number of scripts per (emulated) user
#                        respectively; then nspu = ns / nu.  If
#                        both -nspu and -nu are specified, -nu
#                        takes precedence.  The default value of nu
#                        is ns.
#
#          -suf file     Name of the script user file. The first
#                        field of sufile is a script name.
#                        Following fields are user names which can
#                        execute that script.  Fields are separated
#                        by blanks or tabs.  Blank lines and
#                        comments (beginning with "#" are ignored.
#
#          -sut host     Hostname or IP number of the system under
#                        test.  The default value of host is
#                        "localhost".
#
#     AUTHOR
#          William A. Ward, Jr., University of South Alabama.
#
#-------------------------------------------------------------------

CMD=`basename $0`
CMD_DIR=`dirname $0`
AWK1="$AWK"
AWK1="$AWK1 -f $CMD_DIR/prerr.awk"
AWK1="$AWK1 -f $CMD_DIR/prmt.awk"
AWK1="$AWK1 -f $CMD_DIR/randij.awk"
AWK1="$AWK1 -f $CMD_DIR/ltab.awk"
AWK1="$AWK1 -f $CMD_DIR/sc2mt.awk"
AWK1="$AWK1 X $*"
eval $AWK1
```

sc2mt.awk

```
#! /bin/nawk -f
#-------------------------------------------------------------------
#
#     NAME
#          sc2mt.awk
#
#     SYNOPSIS
#          [awk -f] sc2mt.awk X [-mode number] [-nspu number]
#               [-nu number] -suf file [-sut host] [file ...]
#
#     DESCRIPTION
#          sc2mt.awk produces a PurePerformix(tm)-compatible mix
#          table using one or more files of script counts.sc2mt.awk
#          is usually invoked from the shell script "wrapper"
```

```
#          sc2mt.
#
#     OPTIONS
#          See sc2mt for a description of the options.
#
#     SEE ALSO
#          sc2mt
#
#     AUTHOR
#          William A. Ward, Jr., University of South Alabama.
#
#-------------------------------------------------------------------

#-------------------------------------------------------------------
#
#     FUNCTION
#          sc2mt_getargs
#
#     PURPOSE
#          To get input arguments off of the command line.
#
#-------------------------------------------------------------------

function sc2mt_getargs( argc,i) {

     #       Set default values for arguments.

     MODE = 1
     NSPU = 1
     NU   = 0
     SUT  = "localhost"

     #       Process the arguments in the array ARGV.

     argc = ARGC
     ARGC = 1
     for ( i=2; i<argc; i++ )
          if ( ARGV[i] == "-mode" )
               MODE = ARGV[++i]
          else if ( ARGV[i] == "-nspu" )
               NSPU = ARGV[++i]
          else if ( ARGV[i] == "-nu" )
               NU = ARGV[++i]
          else if ( ARGV[i] == "-suf" )
               SUF = ARGV[++i]
          else if ( ARGV[i] == "-sut" )
               SUT = ARGV[++i]
          else if ( substr(ARGV[i],1,1) == "-" && length(ARGV[i])
          > 1 )
               prerr( "sc2mt: Invalid argument " ARGV[i])
          else
               ARGV[ARGC++] = ARGV[i]

     #       Check for errors in the input arguments.

     if ( MODE < 1 )
          prerr( "sc2mt: Argument mode is invalid" )
     if ( NSPU < 1 )
          prerr( "sc2mt: Argument nspu is invalid" )
     if ( NU < 0 )
          prerr( "sc2mt: Argument nu is invalid" )
     if ( SUF == "" )
          prerr( "sc2mt: Required argument suf not supplied" )
}

#-------------------------------------------------------------------
```

```
#
#     FUNCTION
#          sc2mt_init
#
#     PURPOSE
#          To check for errors in the input arguments.
#
#-----------------------------------------------------------------

function sc2mt_init(    pid) {
     MAX_SEQ = 999999999
     NS      = 0
     "echo $$" | getline pid
     TMP_FILE = "/tmp/sc2mt." pid
     TMP_FMT  = "%s %s %9.9d\n"
     system( "rm -f " TMP_FILE )
}

#-----------------------------------------------------------------
#
#     FUNCTION
#          sc2mt_mode1
#
#     PURPOSE
#          Executed for every input line when in mode 1.  In
#          mode 2, the number of mix table entries for this script
#          is the script count ($2) from the input file.
#
#-----------------------------------------------------------------

function sc2mt_mode1(   j,n,script,seqn,user) {
     script = $1
     n = $2
     for ( j=1 ; j<=n ; j++ ) {
          NS   = NS + 1
          user = ltab_getval( SU_LTAB, script, "RANDOM" )
          seqn = randij( 1, MAX_SEQ )
          printf TMP_FMT, script, user, seqn >> TMP_FILE
     }
}

#-----------------------------------------------------------------
#
#     FUNCTION
#          sc2mt_mode2
#
#     PURPOSE
#          Executed for every input line when in mode 2.  In
#          mode 2, there is one mix table entry for each valid
#          script user and the script execution order is mot
#          randomized.
#
#-----------------------------------------------------------------

function sc2mt_mode2(   j,n,script,seqn,user) {
     script = $1
     n = ltab_getval( SU_LTAB, script, "NCOLS" )
     for( j=1 ; j<=n ; j++ ) {
          NS   = NS + 1
          user = ltab_getval( SU_LTAB, script, j )
          seqn = randij( 1, MAX_SEQ )
          printf TMP_FMT, script, user, seqn >> TMP_FILE
     }
}

#-----------------------------------------------------------------
#
```

```
#     FUNCTION
#          sc2mt_end
#
#     PURPOSE
#          Executed once inside END when MODE = 2.
#
#-------------------------------------------------------------------

function sc2mt_end(     command,i_script,port,rspu) {
     #       If specified, -nu overrides -nspu
     #       rspu is number of remaining extra scripts after
     #       NSPU scripts have been assigned to each rte user.

     if ( NU > 0 ) {
          NSPU = int( NS / NU ) + 1
          rspu = NS % NU
     } else {
          NU   = int( NS / NSPU )
          rspu = -1
     }

     #  i_script counts scripts for a single rte user.

     i_script = NSPU + 1

     #  Sort the temp file by sequence number & read from it until
     eof.

     port    = "telnet:" SUT
     command = "sort -n -k 3 " TMP_FILE

     while ( command | getline ) {

          #   If enough scripts have been entered in the mix table
          #   for this rte user, correct NSPU if the extra scripts
          #   are gone and set the counters for the next rte user.

          if ( i_script > NSPU ) {
               i_script = 1
               prmt_reset()
               if ( rspu == 0 )
                    NSPU = NSPU - 1
               rspu = rspu - 1
          }

          #       Write the next line in the mix table.

          prmt_line($1,port,$2,"t")
          i_script = i_script + 1
     }

     #       Remove the temporary file.

     system( "rm -f " TMP_FILE )
}

#-------------------------------------------------------------------
#
#     ROUTINE
#          Main program
#
#     PURPOSE
#          Top level procedure invocations.
#
#-------------------------------------------------------------------
```

```
BEGIN {                              # Initialization.
     sc2mt_getargs()                 # Process command-line arguments.
     sc2mt_init()                    # Initialize variables.
     ltab_build( SU_LTAB, SUF )      # Build the script-user list.
}
{                                    # Repeated for each input line.
     if ( MODE == 1 )
          sc2mt_mode1()
     else
          sc2mt_mode2()
}
END {                                # Wrap up at end.
     if ( error )
          exit
     sc2mt_end()
}
```

tc2sc

```
#! /bin/sh
#-----------------------------------------------------------------
#
#    NAME
#         tc2sc (Transaction Count To Script Count)
#
#    SYNOPSIS
#         tc2sc [-ed yyyymdd] [-hpd number] [-hpt number]
#              [-iw] [-ot type] [-pf number] [-sff file] [file
#              ...]
#
#    DESCRIPTION
#         tc2sc.awk produces script counts using one or more files
#         of transaction counts. tc2sc is a shell script "wrapper"
#         for tc2sc.awk.
#
#    OPTIONS
#         -ed yyyymmdd  Exclude date from calculations; may appear
#                       multiple times.
#
#         -hpd number   Hours per day.  Default is 24.
#
#         -hpt number   Hours per test. Default is 1.
#
#         -iw           Include weekend days in calculations.
#                       Default is to not include weekends.
#
#         -ot type      Output type; legal values are "long",
#                       "short","avg", "1sd", "2sd" (default), and
#                       "max".
#
#         -pf number    Peaking factor.  Default value is 1.
#
#         -sff file     Scale factor file.  Scale factors are set
#                       to 1 if file not specified.
#
#         file ...      Script names are assumed to be the names
#                       of these files, less the suffix.
#
#    SEE ALSO
#         tc2sc.awk
#
#    AUTHOR
#         William A. Ward, Jr., University of South Alabama.
#
#-----------------------------------------------------------------
```

```
CMD=`basename $0`
CMD_DIR=`dirname $0`
AWK1="$AWK"
AWK1="$AWK1 -f $CMD_DIR/date.awk"
AWK1="$AWK1 -f $CMD_DIR/fmthdg.awk"
AWK1="$AWK1 -f $CMD_DIR/insert.awk"
AWK1="$AWK1 -f $CMD_DIR/prerr.awk"
AWK1="$AWK1 -f $CMD_DIR/prttab.awk"
AWK1="$AWK1 -f $CMD_DIR/tc2sc.awk"
AWK1="$AWK1 X $*"
eval $AWK1
```

tc2sc.awk

```
#! /bin/nawk -f
#--------------------------------------------------------------------
#
#    NAME
#         tc2sc.awk
#
#    SYNOPSIS
#         [nawk -f] tc2sc.awk X [-ed yyyymdd][-hpd number][-hpt
#         number]
#              [-iw] [-ot type] [-pf number] [-sff file] [file
#              ...]
#
#    DESCRIPTION
#         tc2sc.awk produces script counts using one or more files
#         of transaction counts.  tc2sc.awk is usually invoked
#         from the shell script "wrapper" tc2sc.
#
#    OPTIONS
#         See tc2sc for a description of the options.
#
#    SEE ALSO
#         tc2sc
#
#    AUTHOR
#         William A. Ward, Jr., University of South Alabama.
#
#--------------------------------------------------------------------

#--------------------------------------------------------------------
#
#    FUNCTION
#         tc2sc_getargs
#
#    PURPOSE
#         To get input arguments off of the command line.
#
#--------------------------------------------------------------------

function tc2sc_getargs( argc,i) {

     #       Set default values for arguments.

     IW = 0
     HPD = 24
     HPT = 1
     OT = "2sd"
     PF = 1.0
     SFF = ""

     #       Process the arguments in the array ARGV.
```

```
        argc = ARGC
        ARGC = 1
        for ( i=2; i<argc; i++ )
             if ( ARGV[i] == "-ed" )
                  ED = ED " " ARGV[++i]
             else if ( ARGV[i] == "-hpd" )
                  HPD = ARGV[++i]
             else if ( ARGV[i] == "-hpt" )
                  HPT = ARGV[++i]
             else if ( ARGV[i] == "-iw" )
                  IW = 1
             else if ( ARGV[i] == "-ot" )
                  OT = ARGV[++i]
             else if ( ARGV[i] == "-pf" )
                  PF = ARGV[++i]
             else if ( ARGV[i] == "-sff" )
                  SFF = ARGV[++i]
             else if ( substr(ARGV[i],1,1) == "-" && length(ARGV[i])
             > 1 )
                  prerr( "tc2sc: Invalid argument " ARGV[i])
             else
                  ARGV[ARGC++] = ARGV[i]

        #       Check for errors in the input arguments.

        if ( HPD < 0 || HPD > 24 )
             prerr( "tc2sc: Argument hpd is invalid" )
        if ( HPT < 0 || HPT > 24 )
             prerr( "tc2sc: Argument hpt is invalid" )
   }

   #-----------------------------------------------------------------
   #
   #    FUNCTION
   #         tc2sc_init
   #
   #    PURPOSE
   #         To check for errors in the input arguments.
   #
   #-----------------------------------------------------------------

   function tc2sc_init() {
        #      Initialize various parameters for calculating
               statistics

        n_typ = 0
        n_ymd = 0
        sf[0] = 0
        typ[0] = ""
        typ_exists[0] = ""
        ymd[0] = ""
        ymd_exists[0] = ""

        #       Initialization for date functions

        date_moty2mm_init()
        date_yyyymmdd2n_init()
        date_n2dotw_init()
   }

   #-----------------------------------------------------------------
   #
   #    FUNCTION
   #         tc2sc_calcsf
   #
   #    PURPOSE
   #         Calculate scale factors for each transaction type.
```

```
#         The default scale factor is the product of the peaking
#         factor and the ratio of the test duration to the
#         length of the day.  This default may be modified
#         using values from the scale factor file.
#
#-------------------------------------------------------------------

function tc2sc_calcsf(peaking_factor,hours_per_day,hours_per_test,
     scale_factor_file,sf,           i,t) {


     t = peaking_factor * date_hm2m( hours_per_test ) \
          / date_hm2m( hours_per_day )
     for ( i in typ_exists )
          sf[i] = t

     if ( scale_factor_file != "" ) {
          while ( getline < scale_factor_file > 0 )
               sf[$1] = sf[$1] * $2
   }
}

#-------------------------------------------------------------------
#
#    FUNCTION
#         tc2sc_
#
#    PURPOSE
#
#-------------------------------------------------------------------

function tc2sc_() {
}

#-------------------------------------------------------------------
#
#    FUNCTION
#         tc2sc_end
#
#    PURPOSE
#
#-------------------------------------------------------------------

function tc2sc_end() {
}

#-------------------------------------------------------------------
#
#    ROUTINE
#         Main program
#
#    PURPOSE
#         Top level procedure invocations.
#
#-------------------------------------------------------------------

BEGIN {                          # Initialization.
     tc2sc_getargs()             # Process command-line arguments.
     tc2sc_init()                # Initialize variables.
}
{
     #     Skip lines for which positions 4-6 are not a month name

     mm = date_moty2mm( substr($1,4,3) )
     if ( mm <= 0 )
          next
```

```
        #       Convert dd-MMM-yy to yyyymmdd format

        yyyymmdd = date_yy2yyyy(substr($1,8,2)) mm substr($1,1,2)

        #   If weekends should be excluded and this is a weekend day,
        #   skip this transaction

        if ( ! IW ) {

        #       If necessary, calculate day of the week from day

        if ( ! ( yyyymmdd in dotw ) )
             dotw[yyyymmdd] = substr( \
                  date_n2dotw( date_yyyymmdd2n(yyyymmdd) ), 1,3)
        if ( dotw[yyyymmdd]=="Sun" || dotw[yyyymmdd]=="Sat" )
             next
        }

        #      If this date should be excluded, skip this transaction

        if ( index( ED, yyyymmdd ) )
             next

        #       Get transaction type from file name

        if (FILENAME != prev_file) {
             trans_type = FILENAME
             gsub( /^.*\//, "", trans_type)       # Delete dir string
             gsub( /\.[^.]*$/, "", trans_type)    # Delete suffix
             prev_file  = FILENAME
        }

        #       Update lists and accumulate total trans count

        n_typ = insert(trans_type,typ,n_typ,typ_exists)
        n_ymd = insert(yyyymmdd,ymd,n_ymd,ymd_exists)
        cnt[trans_type,yyyymmdd] = cnt[trans_type,yyyymmdd] + $2
}

END {
        #       Initialize various parameters for displaying results

        j_grp = 10
        cnt_fmt_wid = 7
        typ_hdg_len = 80 - j_grp * cnt_fmt_wid - 2
        cnt["FORMAT"]     = "%" cnt_fmt_wid "d"
        typ_hdg["FORMAT"] = "%-" typ_hdg_len "s "
        ymd_hdg["FORMAT"] = "%" cnt_fmt_wid "s"
        sc_fmt = "%-8s %" cnt_fmt_wid "d\n"

        tc2sc_calcsf(PF,HPD,HPT,SFF,sf)

        ymd[n_ymd+1] = "total";
        ymd[n_ymd+2] = "avg";
        ymd[n_ymd+3] = "std dev";
        ymd[n_ymd+4] = "max";
        ymd[n_ymd+5] = "avg test";
        ymd[n_ymd+6] = "1sd test";
        ymd[n_ymd+7] = "2sd test";
        ymd[n_ymd+8] = "max test";
        j_end = n_ymd + 8;

        #       Calculate statistics for each trans type

        for ( i=1; i<=n_typ; i=i+1 ) {
             sum = 0;
             ssq = 0;
```

```
     max = 0;
     for ( j=1; j<=n_ymd; j=j+1 ) {
          cij = cnt[typ[i],ymd[j]];
          sum = sum + cij;
          ssq = ssq + cij*cij;
          if ( cij > max )
               max = cij;
     }
     avg = sum / n_ymd;
     dev = sqrt( (ssq - sum*sum/n_ymd) / (n_ymd-1) );
     cnt[typ[i],"total"] = sum;
     cnt[typ[i],"avg"] = avg;
     cnt[typ[i],"std dev"] = dev;
     cnt[typ[i],"max"] = max;
     cnt[typ[i],"avg test"] = sf[typ[i]] * avg;
     cnt[typ[i],"1sd test"] = sf[typ[i]] * (avg + dev);
     cnt[typ[i],"2sd test"] = sf[typ[i]] * (avg + 2*dev);
     cnt[typ[i],"max test"] = sf[typ[i]] * max;
}

#       Calculate total trans count for each day
#       and find day with max trans count

jmax = 1;
for ( j=1; j<=j_end; j=j+1 ) {
     sum = 0;
     for ( i=1; i<=n_typ; i=i+1)
          sum = sum + cnt[typ[i],ymd[j]];
     cnt["total",ymd[j]] = sum;
     if ( j <= n_ymd && sum > cnt["total",ymd[jmax]] )
               jmax = j;
}
j_end = j_end + 1;
ymd[j_end] = "day test";

#   Calculate iterations of each trans type
#   for a test period of the specified length
#   which corresponds to the day with the maximum trans count

sum = 0;
for ( i=1; i<=n_typ; i=i+1 ) {
     cnt[typ[i],ymd[j_end]] = sf[typ[i]] *
     cnt[typ[i],ymd[jmax]];
     sum = sum + cnt[typ[i],ymd[j_end]];
}
cnt["total",ymd[j_end]] = sum;
typ[n_typ+1] = "total";
n_row = n_typ + 1;

#  If the output type is "long" or "short"
#  then set the starting point in the array to begin printing
#  else print scenario counts for each type and exit

if ( OT == "long" )
     j_beg = 1
else if ( OT == "short" )
     j_beg = n_ymd + 1;
else {
     j = OT " test"
     for ( i=1; i<=n_typ; i=i+1 )
          printf sc_fmt, typ[i], cnt[typ[i],j] + 0.5
     exit
}

for ( i=1; i<=n_row; i=i+1 )
     typ_hdg[i] = substr( typ[i], 1, typ_hdg_len );
```

```
    for ( j=1; j<=n_ymd; j=j+1 )
        ymd_str[j] = substr(ymd[j],1,4) \
            " " substr(ymd[j],5,2) \
            " " substr(ymd[j],7,2) \
            " " substr( date_n2dotw( date_yyyymmdd2n(ymd[j]) ),
            1,3)
    for ( j=n_ymd+1; j<=j_end; j=j+1 )
            ymd_str[j] = ymd[j];

    max_hdg = fmthdg(ymd_str,j_beg,j_end,ymd_hdg);
    row_brk["total"] = 0;
    col_brk["total"] = 0;

    #       Loop over column groups

    prttab(cnt,typ,ymd,typ_hdg,ymd_hdg,row_brk,col_brk, \
        1,n_row,j_beg,j_end,j_grp,max_hdg)
}
```

wt2pf

```
#! /bin/sh
#-----------------------------------------------------------------
--------------
#
#    NAME
#         wt2pf (WTmpx To Peaking Factor)
#
#    SYNOPSIS
#         wt2pf [-a h[:m]] [-b h[:m]] [-w h[:m]] [file ...]
#
#    DESCRIPTION
#         Process a wtmpx file (containing user login data) to
#         produce a peaking factor.  Unlike pa2pf, no delta value
#         may be supplied; the bucket size is always 1 minute.
#
#    OPTIONS
#         -a h[:m]  Start time in hours (:minutes); input data
#                   prior to this is ignored.  The default is
#                   00:00.
#
#         -b h[:m]  Stop time in hours (:minutes); input data
#                   after this is ignored.  The default is 24:00.
#
#         -w h[:m]  Time window in hours (:minutes); candidate
#                   peaking factors for each possible window of
#                   this size are calculated to determine the
#                   maximum (i.e., windows are (a,a+w),
#                   (a+d,a+d+w), (a+2d,a+2d+w) ..., (b-w,b)).  The
#                   default is 01:00.
#
#    OPERANDS
#         file ...  Path name of one or more input files.  If no
#                   files are specified, the standard input will
#                   be read.
#
#    SEE ALSO
#         last, pa2pf
#
#    AUTHOR
#         William A. Ward, Jr., University of South Alabama.
#
#-----------------------------------------------------------------

CMD=`basename $0`
```

```
CMD_DIR=`dirname $0`
AWK1="$AWK"
AWK1="$AWK1 -f $CMD_DIR/prerr.awk"
AWK1="$AWK1 -f $CMD_DIR/date.awk"
AWK1="$AWK1 -f $CMD_DIR/wt2pf.awk"
AWK1="$AWK1 X $*"
eval $AWK1
```

wt2pf.awk

```
#! /bin/nawk -f
#--------------------------------------------------------------------
#
#    NAME
#         wt2pf.awk
#
#    SYNOPSIS
#         [awk -f] wt2pf.awk X [-a h[:m]] [-b h[:m]] [-w h[:m]]
#         [file ...]
#
#    DESCRIPTION
#         wt2pf.awk processes a wtmpx file to produce a peaking
          factor.
#         wt2pf.awk is usually invoked from the shell script
          "wrapper"
#         wt2pf.
#
#    OPTIONS
#         See wt2pf for a description of the options.
#
#    SEE ALSO
#         last, pa2pf, wt2pf
#
#    AUTHOR
#         William A. Ward, Jr., University of South Alabama.
#
#--------------------------------------------------------------------

#--------------------------------------------------------------------
#
#    FUNCTION
#         wt2pf_getargs
#
#    PURPOSE
#         To get input arguments off of the command line.
#
#--------------------------------------------------------------------

function wt2pf_getargs( argc,i) {

     #       Set default values for arguments.

     A  = "00:00"
     B  = "24:00"
     D  = "00:01"
     W  = "01:00"

     #       Process the arguments in the array ARGV.

     argc = ARGC
     ARGC = 1
     for ( i=2; i<argc; i++ )
          if ( ARGV[i] == "-a" )
               A = ARGV[++i]
```

```
            else if ( ARGV[i] == "-b" )
                 B = ARGV[++i]
            else if ( ARGV[i] == "-w" )
                 W = ARGV[++i]
            else if ( substr(ARGV[i],1,1) == "-" && length(ARGV[i])
            > 1 )
                 prerr( "wt2pf: Invalid argument " ARGV[i])
            else
                 ARGV[ARGC++] = ARGV[i]

       #       Argument postprocessing

       TA = date_hm2m(A)
       TB = date_hm2m(B)
       TW = date_hm2m(W)

       #       Check for errors in the input arguments.

       if ( TA >= TB )
            prerr( "wt2pf: Start time >= stop time" )
       if ( TW > TB-TA )
            prerr( "wt2pf: Window size > interval" )
  }

  #-------------------------------------------------------------------
  #
  #    FUNCTION
  #         wt2pf_init
  #
  #    PURPOSE
  #         To check for errors in the input arguments.
  #
  #-------------------------------------------------------------------

  function wt2pf_init(    i) {
  }

  #-------------------------------------------------------------------
  #
  #    FUNCTION
  #         wt2pf_main
  #
  #    PURPOSE
  #         Executed for every input line.
  #
  #-------------------------------------------------------------------

  function wt2pf_main(    ) {

       #       Ignore the last two lines of the wtmpx file.

       if ( $1=="" || $1=="wtmp" ) {
            #       Do nothing.

       #       Pseudouser "reboot" is handled explicitly.

       } else if ( $1 == "reboot" ) {
            i = date_hm2m( $7 )
            N_USERS[i] = "reboot"

       #       Field 1 is a valid login ID.

       } else {
            #       A blank in column 24 indicates that the terminal
            #       for the login session was unknown, and so the
            #       login time is in 1d numbers are one less.
```

```
            if ( substr( $0, 24, 1 ) == " " )
                 i = 6
            else
                 i = 7
            m = date_hm2m( $i )
            N_USERS[m] = N_USERS[m] + 1

            #       Unless the user was still logged when the wtmpx
            #       file was written or was logged out because the
            #       system went down, print an output record for the
            #       user logout

            if ( $(i+2) != "logged" && $(i+2) != "down" ) {
                 m = date_hm2m( $(i+2) )
                 N_USERS[m] = N_USERS[m] - 1
            }
       }
}

#-------------------------------------------------------------------
#
#    FUNCTION
#         wt2pf_end
#
#    PURPOSE
#         Executed at end of program.
#
#-------------------------------------------------------------------

function wt2pf_end(      i,n) {
       for ( i=0; i<1440; i++ ) {
            if ( N_USERS[i]=="reboot" )
                 N_USERS[i] = 0
            else
                 N_USERS[i] = N_USERS[i] + N_USERS[i-1]
       }

       ia = int( TA )
       ib = int( TB )
       iw = int( TW )
       for ( i=ia; i<=ia+iw-1; i++ )
            area = area + N_USERS[i]
       wmax = area
       wcur = area
       for ( i=ia+iw; i<=ib; i++ )
            area = area + N_USERS[i]
       for ( i=ia; i<=ib-iw; i++) {
            wcur = wcur - N_USERS[i] + N_USERS[i+iw]
            if ( wcur > wmax )
                 wmax = wcur
       }
       wavg = area / (TB - TA)
       wmax = wmax / TW
       pf = wmax / wavg
       printf "%9.3f %9.3f %8.3f\n", wmax, wavg, pf
}

#-------------------------------------------------------------------
#
#    ROUTINE
#         Main program.
#
#    PURPOSE
#         Top level procedure invocations.
#
#-------------------------------------------------------------------
```

```
BEGIN {                          # Initialization.
     wt2pf_getargs()             # Process command-line arguments.
     wt2pf_init()                # Initialize variables.
}
{                                # Performed once for each input
line.
     wt2pf_main()
}
END {                            # Wrap up at end.
     if ( error )
          exit
     wt2pf_end()
}
```

Appendix B
Included Files

This appendix contains C header files which are included in the scripts. They served to speed script development and promote script uniformity.

common_decl.h

```
/*--------------------------------------------------------------*/
/*                                                              */
/*   NAME                                                       */
/*        common_decl.h                                         */
/*                                                              */
/*   DESCRIPTION                                                */
/*        Contains preprocessor definitions and declarations    */
/*        common to all scripts.                                */
/*                                                              */
/*--------------------------------------------------------------*/

/*   Define standard prompt string.  */

#define PROMPT          "}% "

/*      Declarations for global variables.      */

int       suspend;       /* 1 to suspend a script, 0 otherwise. */
double    think_high;    /* Upper bound on randomly distributed
                          think times. */
double    think_low;     /* Lower bound on randomly distributed
                          think times. */
int       time_out;      /* Scripts terminate after this many
                          seconds. */
int       type_rate;     /* Type rate in characters per second. */

/*  The following variables are used for processing input
arguments. */

char  argv2[40];         /* Log file name - used for invcreat */
char  argv4[40];         /* Input file name. */
char  argv5[40];         /* Output file name. */
char  input[10][40];     /* Input arguments read from the input
                          file. */

/*      Miscellaneous variables. */
```

```
char  command[40];        /* Used by common_input to test for file
                          existence. */
int i, j, k, l, m, n;     /* Miscellaneous variables for loops. */
int viscreat=0;           /* Used by common_print to signal running
                          viscreat. */
char prc_number[40];      /* Should use one number variable for all
                          of these. */
char tor_number[40];
char cvo_number[40];
char inv_number[5];       /* Invoice number used in INVCREAT */
```

common_gvread.h

```
/*-----------------------------------------------------------------*/
/*                                                                 */
/*   NAME                                                          */
/*        common_gvread.h                                          */
/*                                                                 */
/*   DESCRIPTION                                                   */
/*        Read global variables and use them to initialize         */
/*        parameters common to all scripts.                        */
/*                                                                 */
/*-----------------------------------------------------------------*/

suspend   = Gv_read("SUSPEND");  /* Read the suspend flag */
Gv_readv("TH", &think_high);     /* Read the maximum think time */
Gv_readv("TL", &think_low);      /* Read the minimum think time */
time_out  = Gv_read("TO");       /* Read the time-out time for
                                 hung scripts */
type_rate = Gv_read("TR");       /* Read the type rate */
```

common_input.h

```
/*-----------------------------------------------------------------*/
/*                                                                 */
/*   NAME                                                          */
/*   common_input.h                                                */
/*                                                                 */
/*   DESCRIPTION                                                   */
/*        All scripts use this code to read their input file.      */
/*        The input file is command line argument four.            */
/*                                                                 */
/*-----------------------------------------------------------------*/

/*   Process command line argument 4 (the input file name)   */

if (argc < 5)
     /*      If argv[4] is absent set argv4 to null.         */

     argv4[0] = '\0';
else {
     /*      Build the file name in argv4 and log it.        */
     strcpy( argv4, argv[4] );
     strcat( argv4, ".d\0" );
     /*      Build the command to test for file existence.   */

     strcpy( command, "test -f \0" );
     strcat( command, argv4 );
     Log( command );

     /*      If the file exists, open it and begin reading.  */
     if ( system( command ) == 0 ) {
```

```
          Fioopen( argv4, "r" );
          i=0;
          do {
               Fioreadline( argv4 );
               strcpy( input[i], FIOBUFFER );
               strcat(input[i], "\0");
               Log( input[i] );
               i++;
          } while( FIOLEN >= 0 );
          Fioclose( argv4 );
     }
}

/*   Process command line argument 5 (the output file name)  */

if (argc < 6)
     argv5[0] = '\0';
else {
     strcpy (argv5, argv[5]);
     strcat (argv5, ".d\0");
}
```

common_login.h

```
/*-----------------------------------------------------------------*/
/*                                                                 */
/*   NAME                                                          */
/*        common_login.h                                           */
/*                                                                 */
/*   DESCRIPTION                                                   */
/*        Execute the standard UNIX login sequence, but            */
/*        take into account idiosyncrasies of CEAP machines.       */
/*                                                                 */
/*-----------------------------------------------------------------*/

/*   Standard login sequence ("login: " printed by system)   */

Rcv("login: ");
Kxmit(argv[3], K_ENTER);
Rcv("Password: ");
Kxmit("please", K_ENTER);
i = Mrcv( PROMPT, "(y or n)? ", "% ", "Login incorrect", "" );

/*   Gracefully exit if password is wrong    */

if ( i == 3 ) exit(0);

/*   Login sequence failed; user reenters "login".           */

j=0;
while ( i == 2 ) {
     if (j++ == 5) exit(0);
     Kxmit("login", K_ENTER);
     Rcv("login: ");
     Kxmit(argv[3], K_ENTER);
     Rcv("Password: ");
     Kxmit("please", K_ENTER);
     i = Mrcv( PROMPT, "(y or n)? ", "% ", "" );
}
/*   System queries for multiple sessions (respond "no").    */

while ( i == 1 ) {
     Kxmit("n", K_ENTER);
     i = Mrcv( PROMPT, "(y or n)? ", "" );
}
```

common_logout.h

```
/*-------------------------------------------------------------*/
/*                                                             */
/*   NAME                                                      */
/*        common_logout.h                                      */
/*                                                             */
/*   DESCRIPTION                                               */
/*   Execute standard logout sequence.                         */
/*                                                             */
/*-------------------------------------------------------------*/

/*   Exit last CEFMS menu    */

Begintransaction("enter");
Kxmit("1", K_ENTER);
/*
;1m^[[2J^O^[[0;44;37;1m^[[?1l^[>^[[m^[[2Jcpc25:/wes/u4imcjch{52}%
*/
Rcv(PROMPT);
Endtransaction("enter");

/*   Logout from host        */

Kxmit("logout", K_ENTER);
/* logout^M^J ^M^J You have now logged off cpc25.... ^M^J ^M^J */
Wait(2);
```

common_rte.h

```
/*-------------------------------------------------------------*/
/*                                                             */
/*   NAME                                                      */
/*        common_rte.h                                         */
/*                                                             */
/*   DESCRIPTION                                               */
/*        These are the standard function calls generated by   */
/*        compose at the beginning of every script.            */
/*                                                             */
/*-------------------------------------------------------------*/

Seed(getpid());                          /* Seed random number
                                         generator */
Set(CDELAY);                             /* Put typing delay between
                                         chars. */
Term(ZOOM, VT220|LINES24|AUTOWRAP);      /* Enable zoom & set term
                                         type. */
Thinkuniform(think_low,think_high);      /* Think delay at every
                                         Xmit() */
Timeout(time_out,EXIT);                  /* What to do if Rcv()
                                         takes too long */
Typerate(type_rate);                     /* Typing delay in CPS */
Unset(NOTIFY);                           /* Don't display warnings
                                         (use Mon) */
```

common_start.h

```
/*-----------------------------------------------------------------*/
/*                                                                 */
/*   NAME                                                          */
/*        common_start.h                                           */
/*                                                                 */
/*   DESCRIPTION                                                   */
/*        Include all the required include files; placed           */
/*        at the beginning of every script.                        */
/*                                                                 */
/*-----------------------------------------------------------------*/

#include "common_gvread.h"
#include "common_rte.h"
#include "common_input.h"
#include "common_login.h"
#include "common_suspend.h"
```

common_suspend.h

```
/*-----------------------------------------------------------------*/
/*                                                                 */
/*   NAME                                                          */
/*        common_suspend.h                                         */
/*                                                                 */
/*   DESCRIPTION                                                   */
/*        Suspend code to be included in every script.             */
/*                                                                 */
/*-----------------------------------------------------------------*/

if ( suspend )
        Suspend();
```

common_tms.h

```
/*-----------------------------------------------------------------*/
/*                                                                 */
/*   NAME                                                          */
/*        common_tms.h                                             */
/*                                                                 */
/*   DESCRIPTION                                                   */
/*        Enter the CEFMS database.  If the attempt fails,         */
/*        retry as long as the UNIX C shell prompt is received.    */
/*                                                                 */
/*-----------------------------------------------------------------*/

/* Kxmit("tms", K_ENTER); */
/* qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqj^[[24;1H^OCount:
*0^[[17;51H */
/* Rcv("t: *0^[[17;51H"); */

/*   Build the "tms" command to enter CEFMS  */

strcpy( command, "tms\0" );
/*if ( strlen(argv[2]) == 14 ) {
     command[3] = ' ';
     command[4] = argv[2][13];
     command[5] = '\0';
}*/
/*   Transmit the command 5 times or as long as it fails.    */
```

```
j = 0;
Begintransaction("tms");
do {
     if ( j++ == 5 ) exit(0);
     Kxmit(command, K_ENTER);
} while (Mrcv("t: *0^[[17;51H", PROMPT, "% ", "") != 0 );
Endtransaction("tms");
```

vt220.h

```
/*-----------------------------------------------------------------*/
/*                                                                 */
/*   NAME                                                          */
/*        vt220.h                                                  */
/*                                                                 */
/*   DESCRIPTION                                                   */
/*        Define keystrokes used when accessing                    */
/*        CEFMS via Vistacom.                                      */
/*                                                                 */
/*-----------------------------------------------------------------*/

/*   Miscellaneous Keys      */

#define K_HOME          "^[[1~"       /* VT220 Home */
#define K_INSER         "^[[2~"       /* VT220, CEFMS Insert/Replace
                                         */
#define K_DELETE        "^[[3~"       /* VT220, CEFMS Delete
                                         Character */
#define K_END           "^[[4~"       /* VT220 End */
#define K_PAGE_UP       "^[[5~"       /* VT220 Page Up */
#define K_PAGE_DOWN     "^[[6~"       /* VT220 Page Down */
#define K_UP            "^[[A"        /* VT220, CEFMS Move Up */
#define K_DOWN          "^[[B"        /* VT220, CEFMS Move Down */
#define K_RIGHT         "^[[C"        /* VT220, CEFMS Move Right */
#define K_LEFT          "^[[D"        /* VT220, CEFMS Move Left */

/*   Control Key Sequences    */

#define K_QUIT          "^|"          /* UNIX QUIT (Term Proc & Dump
                                         Core) */
#define K_CTRL_A        "^A"          /* CEFMS Insert/Replace */
#define K_CTRL_B        "^B"          /* CEFMS Beginning of Line */
#define K_INTR          "^C"          /* UNIX INTR (Terminate
                                         Process) */
#define K_EOF           "^D"          /* UNIX EOF, CEFMS Delete
                                         Character */
#define K_CTRL_E        "^E"          /* Unknown Purpose K_CTRL_E */
#define K_CTRL_F        "^F"          /* CEFMS Duplicate Field */
#define K_CTRL_G        "^G"          /* Unknown Purpose K_CTRL_G */
#define K_ERASE         "^H"          /* UNIX ERASE, CEFMS Delete
                                         Backward */
#define K_TAB           "^I"          /* UNIX Tab, CEFMS Next Field
                                         */
#define K_CTRL_J        "^J"          /* UNIX Line Feed */
#define K_CTRL_K        "^K"          /* CEFMS Next Primary Key Field
                                         */
#define K_CTRL_L        "^L"          /* CEFMS Redisplay Page */
#define K_ENTER         "^M"
#define K_CTRL_N        "^N"          /* Unknown Purpose K_CTRL_N */
#define K_DISCARD       "^O"          /* UNIX DISCARD (Discard
                                         Output) */
#define K_CTRL_P        "^P"          /* Unknown Purpose K_CTRL_P */
#define K_START         "^Q"          /* UNIX START (Resume Output)
                                         */
```

```
#define K_REPRINT       "^R"            /* UNIX REPRINT, CEFMS
                                        Redisplay Page */
#define K_STOP          "^S"            /* UNIX STOP (Suspend Output)
                                        */
#define K_CTRL_T        "^T"            /* Unknown Purpose K_CTRL_T */
#define K_KILL          "^U"            /* UNIX KILL (Delete Line) */
#define K_LNEXT         "^V"            /* UNIX LNEXT (Ignore Spec Next
                                        Char)*/
#define K_WERASE        "^W"            /* UNIX WERASE (Word Erase) */
#define K_CTRL_X        "^X"            /* Unknown Purpose K_CTRL_X */
#define K_DSUSP         "^Y"            /* UNIX DSUSP (Delayed Susp FG
                                        Proc) */
#define K_SUSP          "^Z"            /* UNIX SUSP (Suspend
                                        Foreground Proc)*/
#define K_CTRL_RIGHT    "^[OP^[[C"      /* CEFMS Scroll Right */
#define K_CTRL_LEFT     "^[OP^[[D"      /* CEFMS Scroll Left */

/*   Control Function Key Sequences  */

#define K_CTRL_F1       "^[1"           /* Unknown Purpose K_CTRL_F1 */
#define K_CTRL_F2       "^[2"           /* Unknown Purpose K_CTRL_F2 */
#define K_CTRL_F3       "^[3"           /* Unknown Purpose K_CTRL_F3 */
#define K_CTRL_F4       "^[4"           /* Unknown Purpose K_CTRL_F4 */
#define K_CTRL_F5       "^[5"           /* Unknown Purpose K_CTRL_F5 */
#define K_CTRL_F6       "^[6"           /* Unknown Purpose K_CTRL_F6 */
#define K_CTRL_F7       "^[7"           /* Unknown Purpose K_CTRL_F7 */
#define K_CTRL_F8       "^[8"           /* Unknown Purpose K_CTRL_F8 */
#define K_CTRL_F9       "^[9"           /* Unknown Purpose K_CTRL_F9 */
#define K_CTRL_F10      "^[0"           /* Unknown Purpose K_CTRL_F10
                                        */
#define K_CTRL_F11      "^[!"           /* Unknown Purpose K_CTRL_F11
                                        */
#define K_CTRL_F12      "^[@"           /* Unknown Purpose K_CTRL_F12
                                        */

/*   Escape Key Sequences    */

#define K_ESC_POUND     "^[#"           /* CEFMS Count Query Hits */
#define K_ESC_R         "^[r"           /* CEFMS Clear Record */
#define K_ESC_TAB       "^[^I"          /* CEFMS Previous Field */
#define K_ESC_ENTER     "^[^M"          /* CEFMS Previous Field */
#define K_ESC_LEFT      "^[^[[D"        /* CEFMS Beginning of Line */
#define K_ESC           "^["            /* VT220 Escape */

/*   Function Key Sequences  */

#define K_F1            "^[OP"          /* CEFMS Help */
#define K_F2            "^[OQ"          /* CEFMS Enter Query */
#define K_F3            "^[OR"          /* CEFMS Execute Query */
#define K_F4            "^[OS"          /* CEFMS List Field Values */
#define K_F5            ""              /* CEFMS Clear Record */
#define K_F6            "^[[17~"        /* CEFMS Clear Field */
#define K_F7            "^[[18~"        /* Unknown Purpose K_F7 */
#define K_F8            "^[[19~"        /* CEFMS Show Function Keys */
#define K_F9            "^[[20~"        /* Unknown Purpose K_F9 */
#define K_F10           "^[[21~"        /* CEFMS Exit or Previous
                                        Screen */

/*   Shift Key Sequences     */

#define K_SHIFT_F1      "^[[23~"        /* CEFMS Count Query Hits */
#define K_SHIFT_F2      "^[[24~"        /* Unknown Purpose K_SHIFT_F2
                                        */
#define K_SHIFT_F3      "^[[25~"        /* Unknown Purpose K_SHIFT_F3
                                        */
#define K_SHIFT_F4      "^[[26~"        /* Unknown Purpose K_SHIFT_F4
                                        */
```

```
#define K_SHIFT_F5    "^[[28~"      /* Unknown Purpose K_SHIFT_F5
                                    */
#define K_SHIFT_F6    "^[[29~"      /* Unknown Purpose K_SHIFT_F6
                                    */
#define K_SHIFT_F7    "^[[31~"      /* CEFMS Duplicate Field */
#define K_SHIFT_F8    "^[[32~"      /* CEFMS Redisplay Page */
#define K_SHIFT_F9    "^[[33~"      /* CEFMS Print */
#define K_SHIFT_F10   "^[[34~"      /* CEFMS Display Error */
#define K_SHIFT_TAB   "^[[Z"        /* CEFMS Previous Field */
```

Appendix C
System Configurations

RTE Computer System: wescs2.wes.army.mil

System manufacturer	Sun Microsystems
System model	Ultra Enterprise 3000
System architecture	Symmetric shared-memory multiprocessor with processors and memory interconnected using a crossbar switch
Processor architecture	UltraSPARC (9-stage pipeline, superscalar)
Clock rate / cycle time	167 MHz / 6 nanoseconds
Number of processors	4 (out of a possible 6)
Cache per processor	Primary instruction cache: 16 Kbytes (on-chip) Primary data cache: 16 Kbytes (on-chip) Secondary cache: 1 Mbyte (off-chip)
Memory size	320 Mbytes (out of a possible 6 Gbytes)
Peripheral interface	20-Mbyte/sec fast/wide SCSI-2
Network interface	10-Mbyte/sec Ethernet
Operating system	SunOS Release 5.5.1 Version Generic_103640-03 (UNIX® System V Release 4.0)
RTE software	PurePerformix/TTY 3.2.2

SUT Computer System: cpc25.usace.army.mil

System manufacturer	Sun Microsystems
System model	Ultra Enterprise 6000
System architecture	Symmetric shared-memory multiprocessor with processors and memory interconnected using a crossbar switch
Processor architecture	UltraSPARC (9-stage pipeline, superscalar)
Clock rate / cycle time	167 MHz / 6 nanoseconds
Number of processors	24 (out of a possible 30)
Cache per processor	Primary instruction cache: 16 Kbytes (on-chip) Primary data cache: 16 Kbytes (on-chip) Secondary cache: 1 Mbyte (off-chip)
Memory size	5120 Mbytes (out of a possible 30 Gbytes)
Peripheral interface	20-Mbyte/sec fast/wide SCSI-2
Mass storage	See below.
Network interface	10-Mbyte/sec Ethernet
Operating system	SunOS Release 5.5.1 Version Generic_103640-08 (UNIX® System V Release 4.0)
Database system	Oracle Version 7.2.3, CEFMS as of 31 July 1996.

SUT Disks and Database Layout

For these BTs, the Oracle database U4CEFMP1 resided on Sun Model 102 SparcStorage Arrays. An SSA is a separate SCSI disk expansion unit that has three drive trays, with each tray holding 10 half-height, single-connector 3.5-in. Disk drives (a total of 30 drives). Each SSA disk drive operates with a spin rate of 7,200 rpm. Each SSA is linked to the SUT through fiber-optic cables. These cables connect to a Fiber Channel Optical Module (PC/OM) mounted on a Fiber Channel Sbus (FC/S) card on the host side and to an PC/OM mounted on the array controller on the SSA side. At the benckmark site, SSAs are associated with a single host. Graphical User Interface (GUI) software provided by Veritas through Sun allows the configuration management of the SSAs.

Tables C1 and C2 provide detailed information on location of database files and disk drive characteristics. The characteristics of the unmirrored, striped drives were the same as those for the mirrored, striped drives. The user home directories were on unmirrored, unstriped drives that were UFS formatted.

Table C1
Arrangement of U4CEFMP1 Database Files

File	Partition	Status
cefms.df1	/cpc25.d22	mirrored
cefms.df2	/cpc25.d25	mirrored
cefms.df3	/cpc25.d72	not mirrored
cefms_ndx.df1	/cpc25.d39	mirrored
cefms_ndx.df1	/cpc25.d39	mirrored
cemis.df1	/cpc25.d22	mirrored
control01.ctl	/cpc25.d33	mirrored
control02.ctl	/cpc25.d49	mirrored
control03.ctl	/cpc25.d37	mirrored
redo01_a.log	/cpc25.d48	mirrored
redo01_b.log	/cpc25.d54	mirrored
redo01_c.log	/cpc25.d21	mirrored
redo02_a.log	/cpc25.d21	mirrored
redo02_b.log	/cpc25.d48	mirrored
redo02_c.log	/cpc25.d54	mirrored
redo03_a.log	/cpc25.d54	mirrored
redo03_b.log	/cpc25.d21	mirrored
redo03_c.log	/cpc25.d48	mirrored
rollspace.df1	/cpc25.d22	mirrored
system01.dbf	/cpc25.d22	mirrored
tempspace.df1	/cpc25.d48	mirrored
tools01.dbf	/cpc25.d33	mirrored
users01.dbf	/cpc25.d33	mirrored
vims.df1	/cpc25.d54	mirrored
wesreport.df1	/cpc25.d54	mirrored

Table C2
Description of Mirrored Drives

Feature	Status
Fielsystem available space	3,891,384 Kbytes
Filesystem type	UFS
Physical drive type	ST32550W SUN2, 1G, Rev. 0418
Fast write enabled	yes
Mirrored	yes
Physical drives per primary striped plex	2
Physical drives per mirror striped plex	2
Disk block size	512 bytes
Blocks per physical drive	4194995
Striped unit size	80
Read policy	based on plex layout

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
June 1999

3. REPORT TYPE AND DATES COVERED
Final report

4. TITLE AND SUBTITLE
Performance Testing of CEFMS

5. FUNDING NUMBERS

6. AUTHOR(S)
William A. Ward, Jr.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Faculty Court West 20
School of Computer and Information Sciences
University of South Alabama
Mobile, Alabama 36688

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Corps of Engineers, Washington, DC 20314-1000;
U.S. Army Engineer Waterways Experiment Station, 3909 Halls Ferry Road, Vicksburg, MS 39180-6199

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
Technical Report ITL-99-2

11. SUPPLEMENTARY NOTES
Available from National Technical Information Service, 5285 Port Royal Road, Springfield, VA 22161.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report describes a series of benchmark tests of the Corps of Engineers Financial Management System (CEFMS). CEFMS is an interactive software system based on the Oracle relational database management sytem. Remote terminal emulation was used to conduct the tests. CEFMS was installed on a Sun SPARCserver 6000, and its performance in running a series of transactions was observed. The report provides a description of how the test workload was formulated and how the benchmark was prepared, as well as a discussion of the results.

14. SUBJECT TERMS
Benchmarking
Computer system performance evaluation
Corps of Engineers Financial Management System (CEFMS)
Remote terminal emulation

15. NUMBER OF PAGES
93

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED		

13. (Concluded).

c. Of the improvement plans tested, the channel and sand trap configuration of Plan 14 appeared to be optimal with respect to all wave conditions from all directions. Navigation conditions in the entrance will be improved, and the plan will have no negative impact on the existing structures or the spit between the south breakwater and the groin.

d. Sediment tracer tests indicated that sediment moving in the predominant northerly direction will deposit in the deepened entrance channel and sand trap area of Plan 14 as desired, and material moving in the southerly direction will deposit in the deepened entrance channel.

e. The -30-ft entrance channel of Plan 15 will result in similar wave conditions for operational and extreme waves as the -40-ft channel of Plan 14, which would be acceptable with regard to entrance conditions and would have no negative impact on the breakwaters and spit area.