AD-A041 824 ROME AIR DEVELOPMENT CENTER GRIFFISS AFB N Y F/G 9/2
UNATTENDED TESTING SESSIONS ON THE HONEYWELL MULTICS COMPUTER,(U)
MAR 77 D M ELEFANTE

UNCLASSIFIED RADC-TR-77-75 NL

1 OF 1
AD
A 041824

END

DATE
FILMED
8-77

RADC-TR-77-75
In-House Report
March 1977

UNATTENDED TESTING SESSIONS ON THE HONEYWELL MULTICS COMPUTER

Donald M. Elefante

ROME AIR DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
GRIFFISS AIR FORCE BASE, NEW YORK 13441

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be available to the general public, including foreign nations.

This report has been reviewed and is approved for publication.

APPROVED: William F. Stinson

WILLIAM F. STINSON, Major, USAF
Chief, R&D Computer Facility
Information Sciences Division

APPROVED: Robert D. Krutz

ROBERT D. KRUTZ, Colonel, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-77-75	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* UNATTENDED TESTING SESSIONS ON THE HONEYWELL MULTICS COMPUTER		5. TYPE OF REPORT & PERIOD COVERED In-House Report 6. PERFORMING ORG. REPORT NUMBER N/A
7. AUTHOR(s) Donald M. Elefante		8. CONTRACT OR GRANT NUMBER(s) N/A
9. PERFORMING ORGANIZATION NAME AND ADDRESS Rome Air Development Center (ISFO) Griffiss AFB NY 13441		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS P.E. 62702F J.O. 55810218
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (ISFO) Griffiss AFB NY 13441		12. REPORT DATE March 1977 13. NUMBER OF PAGES 38
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)* Same		15. SECURITY CLASS. *(of this report)* UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

Same

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Computers	Digital Computers
Computer Applications	Executive Routines
Executive Systems	System Evaluation
Software Systems	Time-Sharing

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This report discusses the procedure used to run a series of machine-dedicated performance evaluation tests without any machine operator intervention, either before or after the tests, and with minimum disruption to normal time-sharing service. The procedure involves, among other things, setting up a control program to execute at some optimum time in the future, whereupon MULTICS is automatically induced to remove itself from its normal user support status, log in a predetermined set of artificial users for the duration of the test,

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

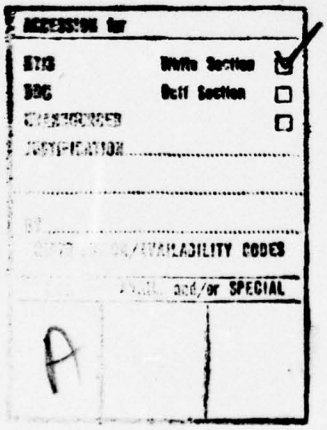
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

and following this, restore itself to its normal user (timesharing) status.

77-75

INTRODUCTION

Multics is a state-of-the-art operating system currently running on several Honeywell Information Systems computers in the 6000 line. It is a very powerful time-sharing system which lends itself to a great deal of hands-off versatility. This report discusses the procedure used to run a series of machine-dedicated performance evaluation tests without any machine operator intervention, either before or after the tests, and with minimum disruption to normal timesharing service. The procedure involves, among other things, setting up a control program to execute at some optimum time in the future, whereupon Multics is automatically induced to remove itself from its normal user support status, log in a predetermined set of artificial users for the duration of the test, and, following this, restore itself to its normal user (timesharing) status. The value of this procedure may be generalized to any instance where dedicated, hands-off service of some kind is desirable.



MULTICS EXEC_COM FACILITY

An exec_com is a macro capability in Multics which allows a user to execute an ascii file containing one or more interactive command lines, i.e., those lines which are normally typed at a terminal when a user is in the timesharing or interactive mode. A number of bells and whistles are available within the exec_com facility which serve to make it quite flexible and powerful, e.g., if-then-else conditionals, go-to's, interrogation of users, etc. These are called control lines. In addition, when an exec_com is called into execution, arguments may be passed to it as part of the calling sequence, and these arguments can be manipulated within the exec_com.

A user may execute an exec_com at any time by typing the command line at his terminal which calls it into action. It is also possible to call an exec_com within an exec_com (and recursive calls are supported).

THE MULTICS ABSENTEE FACILITY

A user may choose to execute an exec_com in absentia. When executed in this way, it is called an "absentee job". The "enter_absentee_request" command, or "ear" for short, is the vehicle for doing this.

When a user submits an absentee request via "ear", Multics responds in one of two ways, depending on whether or not he wants to defer execution of the exec_com to some later time. If a user does not request deferral of execution, and Multics currently has no restrictions (see below) on queueing or honoring such requests, the absentee job will immediately log in (with the submitting user's ID) and execute. This will be followed by a logout upon completion of the job. If, on the other hand, the user requests execution deferral, Multics will delay execution of the job until arrival of the deferral time specified by him in his "ear" command. When this time finally arrives, Multics immediately executes the job as above, but once again, only if there are no restrictions on queueing or honoring the request. If, for some reason, Multics cannot honor the request when the deferral time arrives, the job will remain in queue until such time as all restrictions are removed. It will then be executed.

The restrictions being referred to above are as follows:

(a) A user must have appropriate permission in order to submit an absentee request. If adequate permission does exist, the request is entered into one of three priority queues, the choice of which is predetermined by the user. If sufficient permission does not exist, the job will never get to the point of being queued.

(b) Since absentee jobs in execution contend with interactive users for computer resources, administrators will often limit the number of absentee jobs allowed to be in execution at any one time. Hence, the queue effect comes into play. Another point of interest is that a user may submit as many absentee jobs as he chooses, and the only thing limiting him from having them all logged in and executing at the same time (other than his deferring execution) are administrative limits placed upon the absentee facility.

(c) An administrator may choose to restrict which of the three absentee priority queues are to be allowed service at any particular time. If, for example, only queue 1, the highest priority queue, is allowed service (queues 2 and 3 being restricted), then a user who has no permission to enter jobs

into queue 1 is effectively prevented from executing any absentee jobs during the restriction period, while another user who has permission to enter jobs into queue 1 has free reign up to the limits imposed by (b).

The absentee user facility in Multics is the backbone of the capability described in this report. Therefore, for emphasis, a summary and enlargement of the absentee facility characteristics (most of which have already been discussed) is now provided:

1) An "absentee" job is analogous to "batch" jobs in conventional operating systems. However, instead of such jobs being executed under the control of a computer operator, an absentee job will either execute almost immediately (upon scheduling), or will be delayed to run no earlier than some predetermined time in the future, all at the discretion of the user who sets up the conditions for running such a job (but see #3).

2) An absentee job is scheduled for running when a user places it into one of several priority queues via the "enter_absentee_request" command (or "ear" for short).

3) An absentee job is actually viewed by Multics as an independent user and, under the control of the Multics absentee user facility, logs into and out of the System in much the same way as a normal user, and with the ID of the user who submits the job. A user has the power to create any number of absentee jobs, but Multics contains the necessary administrative handles to restrict the number of absentee jobs allowed to be executed in the System at any one time, and to restrict priority queues in which these jobs may be scheduled for execution.

4) An absentee job has the same power to perform functions in Multics as the user who enters the request for an absentee job to run. This occurs because an absentee job is nothing more than a stack of interactive user command lines interspersed with appropriate control lines. These command and control lines comprise what is called the absentee exec_com or control file, more commonly referred to as the "absin" file, which is simply an ascii file residing in a Multics user's workspace.

5) When an absentee request is entered by a user, among the options which he has available to him are the following:

a) Specification of the output file (commonly referred to as the "absout" file) to which all output from the absentee job will be directed. This output is equivalent to that

which a normal user receives at his terminal upon executing various command lines. If an output file is not specified, a default file is assigned by Multics.

b) Specification that the job is to be restarted in the event that it is interrupted by the system. This is a complete restart.

c) Specification that a CPU time limit is to be placed on the job.

d) Specification that the job is to be entered into a specific priority queue. If the user has access to that queue, the job will be scheduled for execution. Queue 3 is the default queue, as opposed to queues 1 and 2 which are of higher priority as far as order of execution of jobs is concerned.

e) Specification that the job is not to log in for execution any earlier than a specified time in the future. If a deferred time is not specified, the job will be scheduled and executed as soon as Multics can honor the request.

f) Specification that specific arguments are to be passed to the absentee job at the beginning of execution. This is similar to passing arguments to subroutines, and any such arguments are included as part of the command which enters the absentee request ("ear").

SCHEDULING PERFORMANCE TESTS

An absentee control program or exec_com is used to supervise and control the entire testing procedure, starting from the point where normal user service is suspended right on through to restoration of the normal service following the dedicated operation. Since the best time to run dedicated tests in the RADC/Multics environment is very early in the the morning, normally around 3AM, it's quite practical to set up for the machine-dedicated execution of a performance test during normal work hours and have execution deferred to a more benign time.

An interactive exec_com called "setup" is used to submit the deferred absentee request and perform some housekeeping operations. In general terms, the exec_com performs the following functions:

(1) Delete some files which may have been left over from execution during a previous test session, and make sure a zero-length (null) status file exists for the upcoming test.

(2) Ask the submitter (see Appendix M) what time he would like to have the test begin. Store his response in a variable called "valu".

(3) Enter an absentee request (called "start_test") into the highest priority queue, queue 1, after the following questions are asked (see Appendix M):

(a) How many processes or artificial users would you like to have logged in for the test? (This has been typically responded to with the number 15)

(b) How many load iterations would you like each artificial user to perform before the test terminates? (Each artificial user serves as an overseer for repeatedly executing a relatively brief program designed to place a load on system resources. The answer to this question determines the number of times each overseer will execute the load program.)

Defer the startup of the test until the time specified in the stored variable "valu". Three arguments will be passed to the control program when it finally begins execution. The arguments are based on the answers to the questions of steps 2 and 3.

(4) Ask the submitter (see Appendix M) what time he would like Multics to restore normal operations if, for some reason, the testing procedure fails to terminate. Store this answer in the variable "valu".

(5) Multics runs with a privileged and very powerful control process called the Initializer System Daemon (typically called the Initializer). In this step, the Initializer is instructed to send itself a delayed, executable memo to the effect that, if the performance test hasn't come to a normal halt, the test is to be aborted and normal operations are to be restored. This memo becomes active or "mature" at the time specified in step 4, and induces the Initializer to perform as indicated. However, if the test comes to a normal conclusion before the memo described herein matures, the system will have already been restored to normal operations and, by virtue of the fact that the memo is recalled or deleted as part of this restoration process, no test-abortion memo will ever get the opportunity to mature.

So, in summary, the setup exec_com performs a little housekeeping, schedules a performance test to run at some time in the future, and caters to the possibility that the test will not terminate properly.

THE GUESTS MUST LEAVE!

When the time arrives for the absentee control program to automatically log in (typically around 3AM at RADC), certain functions must be performed by the control program before the machine can be considered dedicated to the performance testing procedure, not the least of which is to get any other users off the system! In general terms, the sequence of events are as follows:

(1) One of the arguments passed to the absentee control program ("start_test") from the "setup" exec_com is the time said control program was scheduled to log in, plus 2 minutes. As soon as the control program does log in, a check is made to see if the actual log-in time is less than the value passed as an argument. If it is not less, something has gone amiss (perhaps the system was down at the time the job was scheduled to take off) and the session is aborted. Recall, if you will, that a deferred absentee job which is restricted from logging in at its scheduled time will be allowed to log in at its first legitimate opportunity.

(2) Set a flag indicating that the test session is now in progress.

(3) Induce the Initializer to prevent any more interactive users from logging into the system.

(4) Warn all current users that the system is to become unavailable in 10 minutes. They are asked to log out by then. Pause for 10 minutes.

(5) Bump or remove all bumpable users who have failed to log out.

(6) Log out all special system-support users, etc. (except the Initializer System Daemon) which serve to perform various system functions like IO, ARPA Network control, etc.

(7) Execute an exec_com which alters a table controlling the maximum number of users allowed on the system in order to permit numbers large enough to accommodate all the absentee users who might possibly be logging in during the test session. The exec_com also restricts absentee queues 2 and 3 from allowing jobs to log in.

(8) Make the altered table become immediately effective.

(9) Query Multics as to all interactive users who may still be left in the system, and place the results of this query into a temporary file. (Two types of users can conceivably be left on the system despite the removal processes of steps 5 and 6 -- absentee users, and interactive users who are not bumpable by the method used to accomplish step 5 because they have special privileges.) Manipulate this temporary file with an editor such that the file will take on the format of an executable editor macro, whereupon execution of the macro will perform the user bumps which couldn't be done in step 5.

(10) Query Multics as to all absentee users who may still be left in the system, and place the query results into a temporary file. Manipulate this temporary file with an editor such that the file will take on the format of an executable editor macro, whereupon execution of the macro will bump all absentee users from the system, except one! Provisions are made to prevent the absentee job, "start_test", which is in the process of performing all this stuff, from being dislodged from the system. Otherwise, the whole process will end very abruptly!

(11) Induce the Initializer to execute an exec_com which will logically detach all phone lines from the system answering service. This will have the effect of making it unnecessary for Multics to entertain ringing telephones during the testing procedure. Although such a load might not be considered very large, it could easily have undesirable effects on test results. The goal is to have test sessions run as free from any outside perturbations as possible.

(12) Prevent the accounting program from making any updates during the testing procedure. This will eliminate yet another undesirable load from the system. At this point, Multics is as free from unwanted intrusion as it can be. All the resources can be dedicated to servicing only the absentee users who will soon be logging into the system.

(13) Execute an exec_com, called "enter_all_abs_req", which will start the performance test rolling.

THE FUNCTION OF "ENTER_ALL_ABS_REQ"

When the exec_com "enter_all_abs_req" is called in step 13 of the previous section, three arguments are passed to it. The first argument, which might be called "TAKEOFF", is the clock time at which "enter_all_abs_req" is actually called in aforesaid step 13, plus two minutes. "TAKEOFF" will be the clock time that the multiple absentee load overseers will simultaneously log in. The second argument, which might be called "N", represents the number of absentee jobs (load overseers) that will be logged in for the duration of the test. This is the typed value that the submitter responds with in step 3a of section "SCHEDULING PERFORMANCE TESTS". The third and last argument, hereupon named "I" (for Iterations), is the number of times each load overseer will repeatedly execute a specific, fixed load routine.

The four operations which take place upon execution of exec_com "enter_all_abs_req" are as follows:

(1) Enter "N" absentee requests, each representing a load overseer. The names of these absentee jobs are "load_overseer_1", "load_overseer_2", . . . , "load_overseer_N", and each job is queued up in absentee queue 1, the highest priority queue. In addition, they are all scheduled to be deferred until "TAKEOFF". Each job is entered with two arguments -- the first argument being the ID of the absentee job itself (load_overseer_5 has the ID "5" associated with it) and the second argument being "I".

(2) A termination overseer is called into action before "TAKEOFF". This program coordinates and communicates with the absentee load overseers through a status file. It is called with two arguments, "N" and "I", and its job is to (a) set a "termination flag" when and only when each and every absentee load overseer has performed "I" or more executions of the load program, and (b) summarize the data passed to the status file by each load overseer. When the absentee load overseers find the termination flag set, it is their signal to store their thruput data in the status file and log out of the system. The termination overseer waits for all load overseers to store their thruput results in the status file before it summarizes the data.

(3) The exec_com "compile_data" is called with the argument "N". It collects (and in one case compresses) data about each of the absentee overseers for subsequent printing (after the system has been restored to its normal operation). It also queues for printing the absentee output file associated with

"start_test". This data is subsequently examined to verify that nothing unusual occurred during a given test session which might invalidate results.

(4) Induce the Initializer to execute an exec_com that restores the system to its normal user-support status.

THE FUNCTION OF THE LOAD OVERSEERS

Except for its ID (see step 1 of previous section), each absentee load overseer is identical to all the others. Each of them calls the same routine (named "load_control") with two arguments, and when this routine has completed its execution, control is returned to the appropriate caller, which then simply logs out of the system.

As an introduction to the actual method used in the performance test to measure total thruput, each load overseer, by virtue of its subordinate program "load_overseer", executes a "load" "I" times. Clock times are taken both immediately before, and directly after each execution of the load, and the difference between the two is added to a running total. Therefore, after "I" executions of the load, the running total contains the amount of real time spent in execution of the load. If this running time total is divided into the number of iterations, "I", which the load was put through, then a value with the dimensions "iterations per minute" (IPM) is derived. The summation of the various IPM's for all "N" load overseers represents the total thruput, in iterations per minute, managed by the Multics hardware/software configuration at the time of the test. This figure can then be used as a benchmark for certain comparisons, depending on what an analyst has in mind and also on the nature of the "load" (which will be discussed later).

The first argument passed to "load_control" (which is shared amongst the "N" load overseers) is the ID number of the load overseer calling it, and the second argument is "I", the number of iterations through which the load will be placed. The load_control program performs the following functions:

(1) After doing a little housekeeping, load_control executes the load by calling the routine "load" (which is also shared amongst the "N" load overseers). However, the real time used to execute this <u>first</u> call is <u>not</u> added to the running total, nor is it considered to be one of the required "I" iterations. This initial call serves only to "prime" all the Multics hardware and software mechanisms for the activity to follow.

(2) Perform "I" executions of the load, keeping a running total of the real time involved.

(3) The status file which the termination overseer uses to communicate with each of the load overseers has "N" slots in it, each of them eight ascii characters in width. The numbered slot which a particular load overseer uses to pass

information to the termination overseer is the same as the load overseer's ID. When step 2 has run to completion, the load control program writes the word "finished" in its appropriate slot in the status file, and when the termination overseer finds the word "finished" in all "N" slots of the status file, it sets the "termination flag".

(4) There is very likely to be a delay between the time a particular load control program writes "finished" in the status file and the time the termination overseer sets the termination flag. There are two reasons for this delay, the first being that the termination overseer only senses the contents of the status file every 30 seconds, and the second being that one or more other load control programs may still be executing one of its "I" load iterations (the load control programs do not run in exact unison, despite the overseers having logged in simultaneously). In any case, results would be distorted if any one or more load control programs just stopped executing, after it had completed its "I" passes, while others were still running. Therefore, in this step, the load control program looks for the setting of the termination flag, and if not found, executes the load once again. This will continue until the termination flag is found to be set, at which time the program proceeds to the next step.

(5) Calculate IPM and write it into the appropriate slot of the status file. Then return control to the absentee overseer (which thereupon logs out).

THE "LOAD"

The load which each of the "N absentee load overseers imposes on the system "I" times is designed by the analyst to exercise those elements of Multics (hardware and/or software) which are to be benchmarked for future comparisons. It is not the purpose of this report to discuss what kind of loads should be used for whatever purposes, since such determinations are far from trivial. But, in general, it can be said that if one anticipates some kind of change in one or more elements of his hardware/software configuration, and it is desirable to determine whether or not that change has a positive or negative effect on system performance, then, by a judicious choice of a "load", he can make "before and after" comparisons.

At RADC, a very simple load program has been used repeatedly which, from experience, seems to predict, with a reasonable degree of accuracy, percentage changes in normal user thruput when changes in hardware configurations are made. That PL/1 program is as follows:

```
load:  proc;

dcl a(1024) fixed bin(35);
dcl flush entry;
dcl (j, k, sum) fixed bin(35);

     call flush;

     do j = 1 to 100;
         do k = 1 to 1024;
              a(k) = 12345;
              sum = sum + a(k);
              end;
         end;

     end load;
```

It is not likely that other Multics sites would find this particular program as useful as has RADC, but the idea of trying to develop one with a great deal of simplicity is aesthetically pleasing.

SUMMARY

This report has discussed how a certain type of performance test may be automated at sites being supported by Honeywell/Multics computer systems. The primary value of this automation is the ability to provide dedicated, unsupervised sessions during shifts which are typically not manned by support personnel, and which have the least impact on users.

It has been shown that the Multics absentee facility, when exercised by a user who has special access to a few privileged Multics commands or routines, can be used to support dedicated sessions with normal users being supported directly preceding and directly following such sessions. Furthermore, the fail-soft design of the procedure discussed herein insures a minimum of lost time should a particular performance test fail to run to completion for some reason.

DETAILS

This section deals with specific details of implementation of the procedure which has been discussed. Seasoned Multics users will be able to understand the details that follow.

The working directory in which the functions discussed in this report are performed is >udd>sa>e>lt. It must contain the following four segments at the outset (setup.ec has 7 added names):

```
load_control
load
termination_overseer
setup.ec
      start_test.absin
      enter_all_abs_req.ec
      compile_data.ec
      set_config_table.ec
      restore_normal_operations.ec
      attach_phone_lines.ec
      detach_phone_lines.ec
```

The segment "setup.ec" actually contains all the exec_coms and absin programs indicated by the multiple names above. The convention of tacking many exec_coms together and then using a "&goto &ec_name" (with appropriate entry labels) was used to simplify maintenance of the software. However, they may equivalently be broken up into eight separate segments and dealt with in that way. In any case, the appendix listings and descriptions of the exec_coms will be handled separately for the purpose of clarity in presenting this report.

Three subdirectories exist below >udd>sa>e>lt. They are (with added names):

```
source
load_overseer_N.absin
     absin
load_overseer_N.absout
     absout
```

The source directory contains the PL/1 source segments for the three object segments "termination_overseer", "load", and "load_control". The absin directory contains 32 load overseer absin segments (numbered 1 through 32), and the absout directory, empty at the beginning of a given session, is used to house the load overseer absentee output segments as they are created.

The declaration limits of the based variable "ipm_status_seg" in the "termination_overseer" and "load_control" programs allow for up to 32 slots in the status segment. In concert with this, 32 absentee input segments exist in the directory "absin" to allow for up to 32 absentee load overseers to be logged in during a particular session. Fifteen are normally used at RADC.

All the load overseer absin segments, identical except for name, contain the following three lines:

```
cwd >udd>sa>e>lt
load_control &1 &2
&quit
```

setup.ec

The exec_com "setup" is listed in Appendix A, and its functional description is covered in the section SCHEDULING PERFORMANCE TESTS. Two comments may be necessary at this point to clarify the exec_com.

First, the active function "valu" is a general purpose PL/1 utility program written at RADC. It allows one to reset an internal static ascii variable (initially null) through the "reset" entry, to print it's value via the "print" entry, and to use the value by using the active function. If one simply types "valu", instructions for usage are given. The program is listed in Appendix B.

Second, the "sac" or "send_admin_command" is used by privileged Multics users to request that the Initializer perform a function in its own domain. The sac command is used extensively in the software described in this report, particularly in conjunction with "sc_command", which is available only to the Initializer. The routine "sc_command" is used by the Initializer process to execute those functions normally typed at the Initializer under the system control mode by operators.

start_test.absin

A general description of the function of "start_test.absin" is given under the section "THE GUESTS MUST LEAVE!", and a listing may be found in Appendix C. Additional clarification of "start_test.absin" follows:

In line 4, a flag is set in the Initializer's process directory to indicate that the test is under way. The purpose for setting

it in the process directory is that it will disappear if the system should crash. It is important that the flag not be present after a crash and reboot, else "restore_normal_operations.ec" will begin executing when it shouldn't (the results of the "If-failure" memo entered in "setup.ec").

In line 8 of start_test.absin, the exec_com "initl" is referenced. It is used to send the Initializer a command which is longer than the "sac" or "send_admin_command" can handle. It is listed here:

```
&command_line off
&input_line off
&attach
qx
r >scl>execute_long_command.ec
2c
sc_command &1 &2 &3 &4 &5 &6 &7 &8 &9 &10 &11 &12 &13 &14  &15
{continuation of previous line} &16 &17 &18
\f
w
q
sac ec execute_long_command
&quit
```

The exec_com "execute_long_command" under >scl contains three lines, the first being "&command_line off", the third being "&quit", and the second being that which is manipulated by initl.ec above.

In line 42 of start_test.absin, the absentee user "Elefante" is deleted from the editor buffer to prevent his being bumped when the file is executed as an editor macro. The exec_coms "set_config_table" and "detach_phone_lines" can be found in appendices D and E, respectively.

An Initializer call to "act_ctl_$act_ctl_noupdate" can be found two lines from the end of start_test.absin. Multics System Release 4.0 provides this entry for the purpose of turning off account charging, but it does not disable the event channel which allows wakeups at fixed intervals (not desirable during test sessions). Also, when accounting is re-enabled via the "act_ctl_$act_ctl_reable" entry, a time discrepancy is calculated which forces accounting to go into a periodic manual update mode for the remainder of the system-up period, and to print out abnormal messages at the Initializer. In order to avoid both of these problems, "act_ctl_" has been modified at RADC, and an ascii comparison of the old source (standard SR4.0 version) with

the new source (modified 4.0 version) is found in Appendix F.

restore_normal_operations.ec

A copy of this exec_com can be found in Appendix G. Line 2 checks to see if the test is still in progress. Normally it will be, but if the system crashes after the test starts, and before normal operations are restored, the segment "test_in_progress" will vanish from the Initializer's process directory upon rebooting. Without this segment in existence, the "if-failure" memo will perform harmlessly, for it is not very wise to have the system restored to normal operations by this exec_com when it is already operating normally!

Line 4 deletes "test_in_progress" from the process directory, and line 5 removes the "if-failure" memo from the Initializer's memo segment (not necessary, but cleaner).

Line 7 calls the RADC-modified version of "act_ctl_" (see last paragraph under the "start_test.absin" heading in this section).

The exec_com "attach_phone_lines" causes Multics to begin answering telephones once again (they were disabled in "setup.absin").

The exec_com "retype_config_table" restores the configuration table in the system installation parameters segment which were modified in "setup.absin". This exec_com is placed under >scl at RADC because it is sometimes used in other situations. A listing is found in Appendix H.

Appendix A (setup.ec)

```
&command_line off
delete start_test.absout
answer yes -bf dl absout>**
delete termination_flag
&if [not [exists segment ipm_status_seg]]
&then create ipm_status_seg
&else truncate ipm_status_seg
valu$reset [response Start_time?]
ear start_test -q 1 -bf -tm [valu]
% -ag [date_time [valu] 2 minutes] [response "# of processes?"]
% [response "# of load iterations?"]
valu$reset [response "If-failure restore time?"]
sac memo -al -tm [valu] -call ec >udd>sa>e>lt>restore_normal_
%operations
&quit
```

"%" used at beginning of line indicates continuation of previous line (but line not broken in actual implementation)

Appendix B (valu.pl1)

```
valu:  proc;

dcl cu_$af_return_arg entry
% (fixed bin, ptr, fixed bin, fixed bin(35));
dcl return_string char(max_length) varying
% based(return_string_ptr);
dcl return_string_ptr ptr;
dcl nargs fixed bin;
dcl cu_$af_arg_count entry (fixed bin, fixed bin(35));
dcl max_length fixed bin;
dcl ap ptr init(null);
dcl code fixed bin(35);
dcl cu_$arg_ptr entry (fixed bin, ptr, fixed bin, fixed bin(35));
dcl input_arg char(132) based(ap);
dcl ioa_ entry options(variable);
dcl null builtin;
dcl valu char(132) varying int static;
dcl valu_length fixed bin int static;

     call cu_$af_arg_count (nargs, code);
     if code ^= 0 then do;
          call ioa_ ("^/Subroutine ""valu"" must be called as an
% active function with no arguments.");
          call ioa_ ("Other entry points are as follows:^/");
          call ioa_ ("^-valu$reset^-Resets the value of an
% internal static character string variable
^3-(initially null) for subsequent active function calls");
          call ioa_ ("^-valu$print^-Prints (enclosed in double
% quotes) the value of the
^3-internal character string variable^/");
          end;

     else do;
          call cu_$af_return_arg (nargs, return_string_ptr,
% max_length, code);
          return_string = substr (valu, 1, valu_length);
          end;
     return;


reset:              entry;

     call cu_$arg_ptr (1, ap, valu_length, code);
     if ap = null then valu_length = 0;
     else valu = substr (input_arg, 1, valu_length);
     return;
```

```
print:              entry;

     call ioa_ ("^/""^va""^/", valu_length, valu);
     return;


     end valu;
```

"%" used at beginning of line indicates continuation of previous line (but line not broken in actual implementation)

Appendix C (start_test.absin)

```
&attach
&if [greater [date_time] "&1"] &then ioa_ "Test took off too
% late.  Abort!"; rl
cwd >udd>sa>e>lt
sac cr >pdd>!zzzzzzzbBBBBBB>test_in_progress
pause 10
sac sc_command word test Special test session in progress
pause 10
ec >udd>sa>e>t>initl w * * System will become unavailable in 10
% minutes.  Please log out by then.
pause 600
sac sc_command bump * *
pause 30
sac ec admin netdown
pause 10
sac sc_command logout *
pause 20
ec set_config_table
sac sc_command maxu auto
pause 10
truncate ot
fo ot;as_who -interactive -long;co
qedx
r ot
1,3d
$d
1,$s/^.............................//
1,$s/^....../&%/
1,$s/%.*$/%e pause 10/
1,$s/^/e sac sc_command bump /
1,$s/%/\c
/
\b0
e pause 10
q
truncate ot
fo ot;as_who -as;co
qedx
r ot
$d
1,$s/ (.*$/%e pause 10/
1,$s/\c./ /
1,$s/^/e sac sc_command abs bump /
gd/Elefante/
1,$s/%/\c
/
```

```
\bØ
e pause 1Ø
q
sac ec >udd>sa>e>lt>detach_phone_lines
pause 5
sac act_ctl_$act_ctl_noupdate
ec enter_all_abs_req [date_time 2 minutes] &2 &3
&quit
```

"%" used at beginning of line indicates continuation of previous line (but line not broken in actual implementation)

Appendix D (set_config_table.ec)

```
&command_line off
&input_line off
&attach
ed_installation_parms >scl>installation_parms
r conf
1 512 0 1 0 41.0 41.0 40 1
1 512 0 2 0 41.0 41.0 40 1
1 512 0 3 0 41.0 41.0 40 1
1 512 0 4 0 41.0 41.0 40 1
1 768 0 1 0 41.0 41.0 40 1
1 768 0 2 0 41.0 41.0 40 1
1 768 0 3 0 41.0 41.0 40 1
1 768 0 4 0 41.0 41.0 40 1
2 512 0 1 0 41.0 41.0 40 1
2 512 0 2 0 41.0 41.0 40 1
2 512 0 3 0 41.0 41.0 40 1
2 512 0 4 0 41.0 41.0 40 1
2 768 0 1 0 41.0 41.0 40 1
2 768 0 2 0 41.0 41.0 40 1
2 768 0 3 0 41.0 41.0 40 1
2 768 0 4 0 41.0 41.0 40 1
0
w
q
&detach
&quit
```

Appendix E (detach_phone_lines.ec)

```
&command_line off
&print detaching telephone lines
sc_command remove net001
sc_command remove net002
sc_command remove net003
sc_command remove net004
sc_command remove net005
sc_command remove net006
sc_command remove net007
sc_command remove net008
sc_command remove net009
sc_command remove net010
sc_command remove net011
sc_command remove net012
sc_command remove net013
sc_command remove net014
sc_command remove net015
sc_command remove net016
sc_command remove net017
sc_command remove net018
sc_command remove net019
sc_command remove net020
sc_command remove tty000
sc_command remove tty001
sc_command remove tty002
sc_command remove tty003
sc_command remove tty004
sc_command remove tty005
sc_command remove tty006
sc_command remove tty007
sc_command remove tty008
sc_command remove tty009
sc_command remove tty010
sc_command remove tty011
sc_command remove tty012
sc_command remove tty013
sc_command remove tty014
sc_command remove tty015
sc_command remove tty016
sc_command remove tty017
sc_command remove tty018
sc_command remove tty201
sc_command remove tty202
sc_command remove tty203
sc_command remove tty205
sc_command remove tty206
```

```
sc_command remove tty207
sc_command remove tty208
sc_command remove tty209
sc_command remove tty210
sc_command remove tty211
sc_command remove tty212
sc_command remove tty213
sc_command remove tty214
sc_command remove tty215
sc_command remove tty216
sc_command remove tty217
sc_command remove tty218
sc_command remove tty600
sc_command remove tty601
sc_command remove tty602
sc_command remove tty605
sc_command remove tty616
sc_command remove tty617
sc_command remove tty620
sc_command remove tty621
&quit
```

Appendix F
(ascii comparisons of modifications to act_ctl_.pl1)
("A" represents old source, "B" new source)

```
B143          timer_manager_$reset_alarm_wakeup entry
% (fixed bin(71)),

Inserted before:
A143          (sys_log_, sys_log_$error_log) entry options
% (variable),



B787                  updatetime = anstbl.current_time;
%                     /* Set time of last update. */
B788                  anstbl.current_time = clock_ ();
%                     /* Read clock. */
B789                  call datebin_ (anstbl.current_time, absda, mm,
% dd, yy, hh, min, sss, wkd, shf);
B790                  anstbl.shift = shf;
%                                          /* Save current shift */
B791                  call datebin_$revert (mm, dd, yy, hh, 0, 0,
% next_update);
B792                  do while (next_update < anstbl.current_time);
%           /* Compute next update time */
B793                        next_update = next_update +
% installation_parms.acct_update*MILLION;
B794                        end;
B795                  call timer_manager_$alarm_wakeup (next_update,
% "00"b, updchn);
B796                  last_update_interval =
% installation_parms.acct_update;
% /* Save for clock check (in case inst_parms changes) */

Inserted before:
A786                  call sys_log_ (1, "act_ctl_: accounting
% enabled.");



B804                  call timer_manager_$reset_alarm_wakeup
% (updchn);  /* destroys wakeup */

Inserted before:
A793                  call sys_log_ (1, "act_ctl_: accounting update
% disabled.");
```

"&" used at beginning of line indicates continuation of previous line (but line not broken in actual implementation)

Appendix G (restore_normal_operations.ec)

```
&command_line off
&if [not [exists segment >pdd>!zzzzzzzbBBBBBB>test_in_progress]]
&then &quit
delete >pdd>!zzzzzzzbBBBBBB>test_in_progress
memo -dl -match restore_normal_operations
ioa_ "^2/restoring normal operations"
act_ctl_$act_ctl_reable
sc_command abs bump * *
ec >udd>sa>e>lt>attach_phone_lines
sc_command word login
sc_command login Network_Daemon SysDaemon nw
sc_command login Network_Server NetAdmin ns
sc_command login Backup SysDaemon bk
sc_command login IO SysDaemon cord
sc_command login IO SysDaemon prta
sc_command login IO SysDaemon puna
pause 30
sc_command r cord coordinator
sc_command r prta driver
sc_command r puna driver
pause 30
sc_command r prta prta
sc_command r puna puna
ec retype_config_table
sc_command maxu auto
answer yes -bf dp -bf >udd>sa>e>lt>absout>dataseg
answer yes -bf dp -bf >udd>sa>e>lt>absout>absout.archive
answer yes -bf dp -bf >udd>sa>e>lt>start_test.absout
&quit
```

"&" used at beginning of line indicates continuation of previous line (but line not broken in actual implementation)

Appendix H (retype_config_table.ec)

```
&command_line off
&attach
ed_installation_parms >scl>installation_parms
r conf
1 512 0 1 0 21.5 21.5 1 1
1 512 0 2 0 21.5 21.5 5 3
1 512 0 3 0 21.5 21.5 5 3
1 768 0 1 0 23.5 23.5 1 1
1 768 0 2 0 23.5 23.5 5 3
1 768 0 3 0 23.5 23.5 5 3
2 512 0 1 0 26.5 26.5 1 1
2 512 0 2 0 26.5 26.5 5 3
2 512 0 3 0 26.5 26.5 5 3
2 768 0 1 0 31.5 31.5 1 1
2 768 0 2 0 31.5 31.5 5 3
2 768 0 3 0 31.5 31.5 5 3
0
w
q
&detach
&quit
```

Appendix I (start_test.absin)

```
scl 500
ear load_overseer_([index_set &2])
% -bf -q 1 -tm "&1" -ag ([index_set &2]) &3
scl
termination_overseer &2 &3
ec compile_data &2
sac ec >udd>sa>e>lt>restore_normal_operations
&quit
```

"%" used at beginning of line indicates continuation of previous line (but line not broken in actual implementation)

Appendix J (compile_data.ec)

```
&attach
cwd >udd>sa>e>lt>absout
scl 500
ac ad absout load_overseer_([index_set &1]).absout
scl
qedx
bt
r <start_test.absout
vd/Thruput for/
b0
r absout.archive
vd/Iterations/
$a

\bt
\f
e fo ot;pcd;co
bc
r ot
1,/^cpu/-1d
/^clok/,$d
b0
a\bc\f
w dataseg
q
cwd <
&quit
```

Appendix K (termination_overseer.pl1)

```
termination_overseer:  proc(ascii_n_loads, ascii_n_iterations);

dcl ascii_n_loads char(*);
dcl ascii_n_iterations char(*);
dcl character_value char(7);
dcl code fixed bin(35);
dcl cv_dec_check_ entry (char(*), fixed bin(35))
% returns (fixed bin(35));
dcl cv_float_ entry (char(*), fixed bin(35), float bin(27));
dcl fpsum float bin(27);
dcl fpval float bin(27);
dcl get_wdir_ entry returns(char(168) aligned);
dcl hcs_$initiate entry (char(*), char(*), char(*), fixed bin(1),
% fixed bin(2), ptr, fixed bin(35));
dcl hcs_$make_seg entry (char(*), char(*), char(*), fixed bin(5),
% ptr, fixed bin(35));
dcl i fixed bin;
dcl ioa_ entry options(variable);
dcl ipm_status_seg(32) char(8) based (status_ptr);
dcl n_loads fixed bin(35);
dcl n_iterations fixed bin(35);
dcl null builtin;
dcl status_ptr ptr;
dcl substr builtin;
dcl termination_flag_ptr ptr;
dcl timer_manager_$sleep entry (fixed bin(71), bit(2));

    n_loads = cv_dec_check_ (ascii_n_loads, code);
     if code ^= 0 then do;
         call ioa_ ("Bad argument 1 input to
% termination_overseer.  Abort.");
         call hcs_$make_seg ((get_wdir_()), "termination_flag",
% "", 10, termination_flag_ptr, code);
         return;
         end;

    n_iterations = cv_dec_check_ (ascii_n_iterations, code);
     if code ^= 0 then do;
         call ioa_ ("Bad argument 2 input to
% termination_overseer.  Abort.");
         call hcs_$make_seg ((get_wdir_()), "termination_flag",
% "", 10, termination_flag_ptr, code);
         return;
         end;

     status_ptr = null;
```

```
     call hcs_$initiate ((get_wdir_()), "ipm_status_seg",
% "", 0, 0, status_ptr, code);
     if status_ptr = null then do;
          call ioa_ ("Unable to initiate ipm_status_seg.
%  Abort.");
          call hcs_$make_seg ((get_wdir_()), "termination_flag",
% "", 10, termination_flag_ptr, code);
          return;
          end;

     do i = 1 to n_loads;
          ipm_status_seg(i) = "ZZZZZZZZ";
          end;

     do i = 1 to n_loads;
          do while (ipm_status_seg(i) = "ZZZZZZZZ");
               call timer_manager_$sleep(30, "11"b);
               end;
          end;

     termination_flag_ptr = null;
     call hcs_$make_seg ((get_wdir_()), "termination_flag", "",
% 10, termination_flag_ptr, code);

     call timer_manager_$sleep(30, "11"b);

     do i = 1 to n_loads;
          do while (ipm_status_seg(i) = "finished");
               call timer_manager_$sleep(30,"11"b);
               end;
          end;
     fpsum = 0.;

     do i = 1 to n_loads;
          character_value = substr(ipm_status_seg(i), 1, 7);
          call cv_float_ (character_value, code, fpval);
          fpsum = fpsum + fpval;
          end;

     call ioa_ ("^3/***** Thruput for ^d loads and ^d iterations
% is ^.4f iterations per minute *****^3/", n_loads, n_iterations,
% fpsum);

end termination_overseer;


"%" used at beginning of line indicates continuation of previous
    line (but line not broken in actual implementation)
```

Appendix L (load_control.pl1)

```
load_control:  proc;

dcl ap ptr;
dcl ascii_load_id char(2) based(ap);
dcl ascii_iteration_cutoff char(4) based(ap);
dcl clock_ entry returns (fixed bin(71));
dcl code fixed bin(35);
dcl cu_$arg_ptr entry (fixed bin, ptr, fixed bin, fixed bin(35));
dcl finish_time fixed bin(71);
dcl fixed builtin;
dcl float builtin;
dcl get_wdir_ entry returns (char(168));
dcl hcs_$initiate entry (char(*), char(*), char(*), fixed bin(1),
% fixed bin(2), ptr, fixed bin(35));
dcl hcs_$make_seg entry (char(*), char(*), char(*), fixed bin(5),
% ptr, fixed bin(35));
dcl ioa_ entry options(variable);
dcl ioa_$rs entry options(variable);
dcl ipm float bin;
dcl ipm_status_seg(32) char(8) based(status_ptr);
dcl iteration_count float bin;
dcl iteration_cutoff float bin;
dcl len fixed bin(17);
dcl load entry;
dcl load_id fixed bin(35);
dcl null builtin;
dcl start_time fixed bin(71);
dcl status_ptr ptr;
dcl substr builtin;
dcl termination_flag_ptr ptr;
dcl total_minutes float bin;
dcl total_time fixed bin(71);

     call cu_$arg_ptr (1, ap, len, code);
     if code ^= 0 then do;
         call ioa_ ("Problem with argument 1.  Abort.");
         call hcs_$make_seg ((get_wdir_()), "termination_flag",
% "", 10, termination_flag_ptr, code);
         return;
         end;

     load_id = fixed(substr(ascii_load_id, 1, len), 35);

     call cu_$arg_ptr (2, ap, len, code);
     if code ^= 0 then do;
         call ioa_ ("Problem with argument 2.  Abort.");
```

```
        call hcs_$make_seg ((get_wdir_()), "termination_flag",
% "", 10, termination_flag_ptr, code);
        return;
        end;

    iteration_cutoff = float(substr(ascii_iteration_cutoff, 1,
% len), 27);

    total_time = 0;
    total_minutes = 0.;
    status_ptr = null;

    call hcs_$initiate ((get_wdir_()), "ipm_status_seg",
% "", 0, 0, status_ptr, code);
    if status_ptr = null then do;
        call ioa_ ("No ipm_status_seg.  Abort.");
        return;
        end;

    call load;

    do iteration_count = 1.0 to iteration_cutoff by 1.0;
        start_time = clock_();
        call load;
        finish_time = clock_();
        total_time = total_time + (finish_time - start_time);
        end;

    ipm_status_seg(load_id) = "finished";
    termination_flag_ptr = null;

    do while (termination_flag_ptr = null);
        call load;
        call hcs_$initiate ((get_wdir_()), "termination_flag",
% "", 0, 0, termination_flag_ptr, code);
        end;

    total_minutes = float(total_time, 27)/60000000.;
    ipm = iteration_cutoff/total_minutes;
    call ioa_$rs("^7.4f", ipm_status_seg(load_id), len, ipm);
    call ioa_ ("Iterations/minute for load ^2d = ^7.4f",
% load_id, ipm);

end load_control;
```

"%" used at beginning of line indicates continuation of previous line (but line not broken in actual implementation)

Appendix M
(Sample terminal session for job setup)

```
ec setup                          (initiates question sequence)
Start_time?   0300.
# of processes?   15
# of load iterations?   50
If-failure restore time?   0600.
```

METRIC SYSTEM

BASE UNITS:

Quantity	Unit	SI Symbol	Formula
length	metre	m	...
mass	kilogram	kg	...
time	second	s	...
electric current	ampere	A	...
thermodynamic temperature	kelvin	K	...
amount of substance	mole	mol	...
luminous intensity	candela	cd	...

SUPPLEMENTARY UNITS:

plane angle	radian	rad	...
solid angle	steradian	sr	...

DERIVED UNITS:

Acceleration	metre per second squared	...	m/s
activity (of a radioactive source)	disintegration per second	...	(disintegration)/s
angular acceleration	radian per second squared	...	rad/s
angular velocity	radian per second	...	rad/s
area	square metre	...	m
density	kilogram per cubic metre	...	kg/m
electric capacitance	farad	F	A·s/V
electrical conductance	siemens	S	A/V
electric field strength	*volt per metre*	...	V/m
electric inductance	henry	H	V·s/A
electric potential difference	volt	V	W/A
electric resistance	ohm		V/A
electromotive force	volt	V	W/A
energy	joule	J	N·m
entropy	joule per kelvin	...	J/K
force	newton	N	kg·m/s
frequency	hertz	Hz	(cycle)/s
illuminance	lux	lx	lm/m
luminance	candela per square metre	...	cd/m
luminous flux	lumen	lm	cd·sr
magnetic field strength	ampere per metre	...	A/m
magnetic flux	weber	Wb	V·s
magnetic flux density	tesla	T	Wb/m
magnetomotive force	ampere	A	...
power	watt	W	J/s
pressure	pascal	Pa	N/m
quantity of electricity	coulomb	C	A·s
quantity of heat	joule	J	N·m
radiant intensity	watt per steradian	...	W/sr
specific heat	joule per kilogram-kelvin	...	J/kg·K
stress	pascal	Pa	N/m
thermal conductivity	watt per metre-kelvin	...	W/m·K
velocity	metre per second	...	m/s
viscosity, dynamic	pascal-second	...	Pa·s
viscosity, kinematic	square metre per second	...	m/s
voltage	volt	V	W/A
volume	cubic metre	...	m
wavenumber	reciprocal metre	...	(wave)/m
work	joule	J	N·m

SI PREFIXES:

Multiplication Factors	Prefix	SI Symbol
$1\,000\,000\,000\,000 = 10^{12}$	tera	T
$1\,000\,000\,000 = 10^{9}$	giga	G
$1\,000\,000 = 10^{6}$	mega	M
$1\,000 = 10^{3}$	*kilo*	k
$100 = 10^{2}$	hecto*	h
$10 = 10^{1}$	deka*	da
$0.1 = 10^{-1}$	deci*	d
$0.01 = 10^{-2}$	centi*	c
$0.001 = 10^{-3}$	milli	m
$0.000\,001 = 10^{-6}$	micro	μ
$0.000\,000\,001 = 10^{-9}$	nano	n
$0.000\,000\,000\,001 = 10^{-12}$	pico	p
$0.000\,000\,000\,000\,001 = 10^{-15}$	femto	f
$0.000\,000\,000\,000\,000\,001 = 10^{-18}$	atto	a

* To be avoided where possible.

MISSION
of
Rome Air Development Center

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C^3) activities, and in the C^3 areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.