Regular Paper

An Automatic Testing Environment for Large-Scale Operating Systems

SHUNJI TANAKA,† YASUFUMI YOSHIZAWA,† HIDENORI UMENO,† NAOKO IKEGAYA,† KAZUYASU MIYAMOTO †† and NOBUYOSHI SUGAMA ††

This paper presents testing methods for large-scale operating systems (OSs), including for I/O error recovery. To reduce the manpower required for OS testing, we developed the automatic OS test driver (OSTD/AUTO) and the I/O error simulator (IOSIM). OSTD/AUTO automates operations in OS testing by using programmed test procedures written in a test procedure description language. The language describes three types of operations: (a) submitting sequential control commands to the OS in response to the messages displayed to the console, (b) detecting abnormalities in the OS, and (c) collecting information for debugging in response to the detection. OSTD/AUTO decreases operation time to about 67% on the average in comparison with manual operations. In addition, IOSIM can simulate errors during I/O processing and repeated I/O errors in response to the system programmers' specification. Automatic OS testing for I/O error recovery can be done by combining IOSIM and OSTD/AUTO.

1. Introduction

Reliability in computer systems is required, because they are used in a widespread area of activities throughout human society. In particular, large-scale computer systems are now dealing with large-scale data processing indispensable in our daily lives. The OS kernel is a critical element in the computer system. When the OS kernel fails, the entire system is likely to fail. To decrease the number of errors in the OS kernel, a great deal of manpower is required to carry out comprehensive OS testing. Therefore, efficient and effective testing environments for OS kernels must be provided.

OS testing accounts for more than one-third of the total development effors which have been made up to now[1),2)] because OS testing processes have been mainly done manually so far. As the scale of an OS grows larger, the number of test cases to confirm the operating normalities in the OS before shipment is increasing more and more. The OS tests consist of test preparation, test execution and program correction (including analysis of test results). It was considered that the proportion of these three phases to each other is about 2:3:5. However, it is difficult to reduce the amount of time required for program correction, because it requires knowledge of the OS to be tested in detail to find out the causes of program errors. Consequently, to reduce the amount of time and effort required for OS development, test execution has to be carried out automatically. By re-using the programmed test procedure, the amount of preparation time required for retesting will be decreased.

To improve the reliability of computer systems, an OS provides a function to recover from I/O errors. To operate this function effectively, the software that performs I/O error recovery must be reliable. However, about one-fourth of OS failures occur in the error handling.[3)] I/O errors must be supplied for the OS to test the I/O error recovery function. It is difficult, however, to generate hardware errors in I/O devices. As a result, tools to simulate I/O errors must be developed.

We developed the automatic OS test driver (OSTD/AUTO) and the I/O error simulator (IOSIM) to satisfy these requirements. OSTD/AUTO is developed based on the Operating

† Systems Development Laboratory, Hitachi, Ltd.
†† Software Development Center, Hitachi, Ltd.

System Test Driver (OSTD).[4] OSTD mainly employs virtual machine technology[5),6)] to enable multiple system programmers to develop and test an OS simultaneously. OSTD provides each TSS terminal with both the OS screen to operate an OS and the OSTD screen to operate a virtual machine.

Multiple OSs are allowed to achieve a simultaneous operation in a virtual machine system (VMS). VMS is mainly used in a large-scale computer system. A virtual machine monitor (VMM) is a program effecting a control to implement a VMS in which multiple logical machines called virtual machines (VMs) seem to exist in a real computer system. VMM also performs scheduling and dispatching of OSs. An OS on a virtual machine is operated from the console. In addition, virtual machine operations such as initial program loading (IPL) of an OS is performed from the console.

To assure the reliability of large-scale OS, the OS is to be tested under large-scale environments similar to the largest customer's computer system. In addition, each system programmer must prepare each testing environment. Because of space and cost limitations, however, preparing multiple large-scale computers for system programmers is almost impossible.

VMS provides various computer system configurations for virtual machines by simulating computer resources. Regardless of the limitations imposed by the real computer configuration, an OS can be tested under a large-scale computer configuration corresponding to the customer's computer system in VMS environment. VMS also enables multiple system programmers to test and debug OSs simultaneously in a single computer system. In addition, an OS on a virtual machine can be monitored by VMM without modification of the OS. As above, VMS is effective for OS testing.

Some research has already been devoted to OS testing methods and I/O error simulation using virtual machine technology as follows.

(1) A virtual machine system (VMS) with a server virtual machine which automates OS operations was developed.[7] The system includes a server virtual machine and multiple virtual machines to operate the OSs to be tested. The server virtual machine receives I/O operation requests for a console from an OS to be tested, investigates the I/O operation requests, and informs the OS of the corresponding operation.

This system, however, does not automate virtual machine operations such as virtual device definition and initial program loading (IPL) operation of the OS to be tested, nor detect operating abnormalities in the OSs to be tested. In addition, it does not automate the operation of multiple OSs on loosely coupled multiprocessor (LCMP) configured virtual machines. The system is unable to perform these functions because the server virtual machine merely automates OS operations. However, these functions are necessary to automate OS testing.

OSTD/AUTO supports automation of OS testing by replacing the manual testing procedures from the console with programmed test procedures. OSTD/AUTO automates virtual machine operations such as virtual device definition and IPL operation of the OS to be tested by monitoring and controlling both the OS screen and the OSTD screen. OS commands and OSTD commands can be submitted automatically to the OS screen and to the OSTD screen respectively. OSTD commands include the 'DEFINE' command to define virtual devices of the virtual machine and the 'XIPL' (eXtended IPL) command to perform the IPL operation of the virtual machine. It is also able to detect operating abnormalities in the OS to be tested by monitoring the execution of the test procedure. Furthermore, it makes automatic operation of LCMP configured virtual machines possible by communicating between multiple test procedures.

(2) A hardware error generator which simulates I/O errors using a virtual machine system (VMS) was also developed.[8] The generator, however, does not simulate either (a) errors during I/O processing or (b) repeated I/O errors because it simulates an I/O error only in response to a real I/O terminate interruption. However, these functions are necessary to test the I/O error recovery functions of the OS to be tested efficiently.

The I/O error simulator (IOSIM) proposed in this paper can generate errors during I/O processing by embedding causes of errors in the channel program, and can generate repeated errors through the use of timer interrupts to VMM.

In Section 2 of this paper, OSTD is explained. OSTD/AUTO and IOSIM are introduced in Section 3 and 4 respectively. Conclusions are presented in Section 5.

2. Operating System Test Driver (OSTD)

The Operating System Test Driver (OSTD) is developed to support testing for large-scale OSs. System programmers need to conduct the entire OS development sequence from their own workstations at any time as needed. Based on the analysis of system programmers' needs, it was determined that the basic structure of OSTD should be a mixture of the VOS3/TSS, an OS to operate a time sharing system (TSS), and a virtual machine system (VMS). Using a TSS connected to a VMS, a conventional OSTD enables users to test and debug an OS interactively with each TSS terminal in a single computer system in the same manner as the TSS enables them to develop and test application programs interactively.

The configuration of OSTD is presented in **Fig. 1**. As shown in the figure, the TSS terminals used by the system programmers are controlled by VOS3/TSS. The virtual machine which runs VOS3/TSS is called the OSTD Control VM. The following functions were developed for the OSTD Control VM.

(1) The single terminal used by the system programmer can be used for three different purposes. The three sets of functions are the VOS3/TSS terminal function, the VMS console function, and the console function for the OS to be tested. These functions are called the multiple virtual console simulation functions. The multiple virtual console simulation functions were designed to control the terminal states described above and to maintain the screen information for each state.

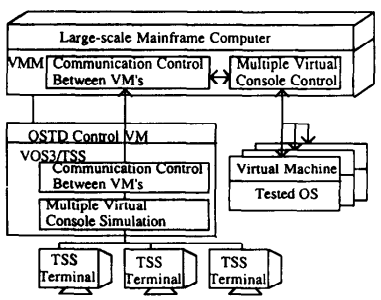

Fig. 1 System configuration of OSTD system.

(2) To collect the VMS console information and the console information for the OS to be tested, communication functions were developed for communicating information between the OSTD Control VM and the VMM.

The following are the principal features of OSTD.

(1) System programmers' workbench

The system programmers' workbench allows programmers to conduct the entire OS development sequences, program creation, compiling, and modification, and OS testing and debugging, at any TSS terminal at any time as needed using both terminal functions and console functions. Using the functions of VMS, multiple OSs can be executed simultaneously, raising the efficiency of computer usage. In addition, the execution of instructions by the OS to be tested can be monitored, halted, and so forth, from the programmers' workbench.

OSTD simulates the console screen of the virtual machine operating the OS to be tested using a TSS terminal under VOS3/TSS. It provides OS screen to submit OS commands and OSTD screen to submit OSTD commands for each TSS terminal. OSTD commands are executed for VOS3/TSS and VMM to assist the OS testing. Virtual machine operations such as initial program loading (IPL) are submitted from OSTD screen by using OSTD commands.

(2) Storage and analysis of OS testing and debugging information

System programmers investigate the causes of program errors by overseeing the OS console operations and analyzing the storage dumps. Functions for collecting and analyzing the storage dumps and for automatically collecting and storing information on OS console operations in a data set are provided.

(3) Efficient, automatic measurement and evaluation of testing thoroughness

System programmers need to know the efficiency of the program testing. The test coverage ratio is useful for determining at what stage testing of an individual module was conducted. A hardware function called Test Coverage Assist (TCA) allows automatic measurement of the thoroughness of testing of program modules.

(4) High level monitoring and control functions for OSs to be tested

In investigating the causes of program errors, system programmers must identify instructions from multiple points which operate in a different manner than intended or which write error data into a program table. Functions are provided for setting multiple control breakpoints within the OS to be tested. In addition, multiple data breakpoints can be set to monitor access to data areas.

(5) Access to data areas using symbolic names

An OS includes a large number of control tables. These control tables are often related in a hierarchical manner. Thus, when accessing storage contents, it is easier to access them not by specifying the memory address, but by defining the relationships among the control tables instead, then specfying a defined symbolic name. Thus, functions to access storage contents by specifying the symbolic name are provided.

(6) Virtual device simulation functions

When testing an OS, it is often necessary to use a very large number of devices and to test the response of the OS to failures in the devices. In addition, testing is often accompanied by manual operation of the devices. For these reasons, multiple virtual terminal simulations (Terminal Traffic Simulator, or TTCS) and other virtual device simulation functions are provided in OSTD.

Implementing the features above, OSTD now enables integrated OS developlent on a single computer system. System programmers did not need to go to the computer room to operate virtual machines for OS testing. Instead, they could perform most of the testing operations interactively from their own TSS terminals at any time, with no need to take over exclusive use of the computer system. Scores of system programmers could perform testing and debugging of OSs simultaneously using OSTD. This was a major breakthrough for testing and debugging of large-scale OSs.

3. Automatic OS Test Driver: OSTD/AUTO

3.1 Requirements for Automatic OS Testing

OSs tend to be developed on the basis of an existing OS. This is especially true for large-scale OSs because they have many general-purpose functions and the scale of them is enormous. Therefore, to confirm the effect on the existing functions in OS development, regression testing accounts for a large part of OS testing. To enable test data to be reused effectively in regression testing and in development of succeeding OSs, tools needed to be developed to execute re-usable programmed test procedures to handle thousands of test cases.

In addition, each system programmer in charge of a function must prepare each testing environment by submitting control commands to test the function. However, much time is required to submit control commands manually for OSs to be tested. To provide a testing environment quickly for system programmers, tools needed to execute automatic computer operations from IPL to any required operation during interactive operations.

In OS testing, system programmers start up for the OS to be tested by submitting control commands and monitor the operating state. If they detect any operating abnormalities in the OS to be tested by console display and so on, they stop the machine and investigate the causes of program errors by checking contents of the general registers (GRs), and memory dumps, etc. This sequence of testing operations must be automated to perform automatic OS testing. This sequence is executed more than one thousand times in a large-scale OS development. In addition, mistakes of test procedures make it necessary to revise the test procedures and do the processing again. In particular, non-interactive processing for an enormous amount of OS retesting and regression testing is likely to be performed at night when no system programmers are present to raise the effciency of computer usage. Consequently, mistakes made in the test procedures in non-interactive processing will cause the confirmation of the test results to be delayed for one day. Therefore, the correctness of the programmed test procedures must be verified before execution. Furthermore, it is difficult to prescribe all the test operations into test procedures to cover unexpected events caused by program errors in OS testing. In addition, test procedures may have to be changed during OS testing. Therefore, it is required to enable intercepting automatic testing to change testing operations during OS testing.

Consequently, the following functions are required for automatic OS testing.

(1) Submitting control commands automatically
(2) Monitoring the operating state of the OS to be tested
(3) Logging during OS testing
(4) Verifying test procedures
(5) Intercepting automatic execution

3.2 Facilities of OSTD/AUTO

OSTD/AUTO is developed based on OSTD. The test procedure operates the OS to be tested in OSTD/AUTO. The system consists of OSTD, testing procedures, procedures interpreter, procedures executer, logging file, and procedures verifier (see **Fig. 2**).

OSTD/AUTO supports automation of OS testing by replacing the manual testing procedures from the console with programmed test procedures. By using OSTD/AUTO, system programmers can make effective use of time during automatic operations. In addition, a lot of regression testing and retesting needs to be carried out in non-interactive processing (what is called "batch" processing) to raise the efficiency of computer usage. Therefore, OSTD/AUTO is designed so that it can be executed both in interactive processing and in non-interactive processing.

Figure 3 shows the state transition involved in switching the TSS mode. When the OSTD command is input while the terminal is in TSS mode, the terminal moves to the OSTD mode and displays the OSTD screen. OSTD commands and TSS commands can be input to the OSTD screen. When the XIPL (eXtended Initial Program Loading) command, one of the OSTD command, is input, the terminal changes to display the OS screen. OS control commands can be input to the OS screen. The switching between the OSTD screen and the OS screen can be done by inputting special key of the terminal. When the 'AUTO START' OSTD command is input, the terminal moves to the OSTD/AUTO mode and the OS to be tested is operated automatically according to the test procedure specified by the command. Mode change from OSTD/AUTO mode to OSTD mode can be done by (a) termination of the test procedure, (b) execution of the 'STOP' statement in the test procedure, and (c) special key.

OSTD/AUTO provides system programmers with the following functions.

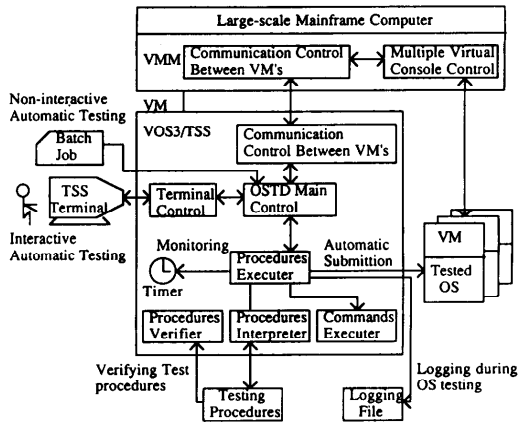

Fig. 2 System configuration of OSTD/AUTO system.

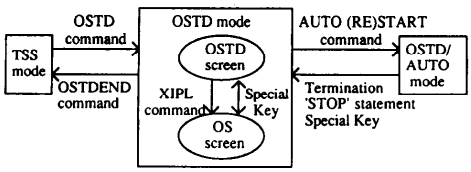

Fig. 3 State transition of the terminal.

(1) Submitting control commands automatically

OSTD/AUTO monitors the message displayed to the OS screen and the OSTD screen according to the 'MATCH' statement in the test procedure, and submits control commands specified by the 'COMMAND' statement. In other words, all messages are ignored until the message specified in the executing 'MATCH' statement is displayed. Therefore, main routine of the testing procedure is a sequence of the messages to be monitored and the control commands to be submitted. The test procedure is usually executed sequentially to operate the OS to be tested. However, console messages from the OS to be tested are sometimes displayed out of turn because of the operating timings. OSTD/AUTO can conduct parallel processing of the test procedure according to the description specified by the 'PARALLEL' statement to submit the control commands corresponding to the console messages out of turn. Thus, OS commands and OSTD commands are submitted automatically by OSTD/AUTO.

Monitoring and controlling of the OS screen and the OSTD screen are performed as follows.

A start I/O instruction issued to the OS console by the OS to be tested in a virtual machine is intercepted and routed to VMM. VMM notifies the contents of the console screen specified by the start I/O instruction to the VOS3/TSS in another virtual machine by using external interruption. Thus, OSTD/AUTO can monitor messages on the OS screen. In the same manner, messages from VMM to OSTD/AUTO are informed to the OSTD screen. Thus, OSTD/AUTO can monitor the OSTD screen. In addition, requests from OSTD/AUTO to VMM such as VMM command execution are informed by using supervisor call (SVC) interruption.

(2) Monitoring the operating state of the OS to be tested

The main routine of the test procedures describes the normal console messages sequence and the normal OS commands sequence. Thus, operating abnormalities in the OS to be tested are detected when the normal message is not displayed to the console. Monitoring such as execution time of the OS to be tested or waiting time for the normal message aids detection of operating abnormalities in the OS to be tested. When an abnormal condition of the OS to be tested specified in the 'ON' statement is detected, the control of the test procedure is transferred to the statement next to the 'ON' statement. After information for debugging is collected by executing the OSTD commands described next to the 'ON' statement, the terminal moves to the OSTD mode.

(3) Logging during OS testing

Commads submitted and messages displayed during the tests are automatically collected in files. Contents of the general registers (GRs) and control registers (CRs) in time of detecting program errors can be collected by submitting the 'DISPLAY' OSTD command in the test procedure. In addition, the memory dumps in time of detecting program errors are collected by submitting the 'DUMP' OSTD command in the test procedure. The 'DUMP' OSTD command can specify the range of real storage and virtual storage. The contents of the dump storage is stored in files of the OS to be tested.

(4) Verifying test procedures

A program to check test-procedure syntax was made to confirm the correctness of test procedures previously.

(5) Intercepting automatic execution

Test procedures can be freely intercepted at specified points by submitting the 'STOP' statement, and the test operations can be executed manually. In addition, the suspended test procedure can be restarted by submitting the 'AUTO RESTART OSTD command.

3.3 Specification of Test Procedure Description Language

Table 1 shows an outline of test procedure description language. The statements shown in the table specify the following actions for OSTD/AUTO.

(1) The 'COMMAND' statement executes a specified control command for either OSTD or the OS according to the specification.

(2) The 'MATCH' statement keeps the execution of the next statement waiting until the specified message is displayed.

(3) The 'ON' statement observes the abnormality according to a specified condition such as the execution time of the test procedure. When an operating abnormality is detected, the control of the test procedure is transferred to the statement next to the 'ON.'

(4) The 'PARALLEL' statement conducts parallel processing of the test procedure to submit the control commands corresponding to the console messages out of turn.

(5) Test procedures can be intercepted by the 'STOP' statement, and the test operations can be executed manually.

(6) 'EXIT' and 'END' statement terminate the test procedure.

Table 1 Specification of test procedure description language.

No.	Statement	Action
1	COMMAND	Executing specified command to either OSTD or the OS
2	MATCH	Keeping specified message waiting
3	ON	Observing the execution of test procedure
4	PARALLEL	Executing processing parallel
5	STOP	Intercepting the execution of test procedure
6	EXIT/END	Terminating the execution of test procedure

3.4 Example of Test Procedure

(1) An example of a test procedure for a single virtual machine

Figure 4 shows an example of a test procedure for a single virtual machine. The test procedure starts an OS and executes the TESTJOB. Furthermore, if the message specified in each 'MATCH' statement is not displayed to the console within ten minutes, OSTD/AUTO displays the contents of general registers (GRs) and the program status word (PSW) and terminates the execution of the test procedure. Consequently, system programmers can investigate the program errors by using the test results.

(2) An example of test procedures for LCMP-configured virtual machines

Figure 5 shows an example of a test procedure for LCMP-configured virtual machines. The test procedure synchronizes and starts two different OSs running on two virtual machines.

(a) The DEFINE command defines a virtual Channel-to-Channel Adapter (CTCA) for each virtual machine. Then the COUPLE command connects the virtual machines with the virtual CTCAs. Consequently, LCMP-configured OSs can communicate with each other using the virtual CTCAs.

```
IPL:PROC();
  COMMAND 'XIPL 440',OSTD;
  MATCH/SPECIFY SYSTEM PARM/;
  COMMAND 'R 00,CLPA',OS;
  MATCH/SPECIFY JSS3 OPTIONS/;
  COMMAND 'R 01,COLD,NOREQ',OS;
  MATCH/JSS3 WAITING FOR WORK/;
  COMMAND 'START TESTJOB',OS;
  EXIT;
  ON TIME(10);
  COMMAND 'SLEEP',OSTD
  COMMAND 'DISPLAY PSW',OSTD
  COMMAND 'DISPLAY GR',OSTD
  END IPL;
```

Fig. 4 A test procedure for a single VM.

(b) Executing XIPL command of the OSTD in two test procedures will perform IPL of OSs for two virtual machines.

(c) After completion of the start-wp for one virtual machine, the Job Spooling Subsystem 4 (JSS4) is loaded into other virtual machine.

(d) System programmers are ready for testing programs in relation to LCMP.

(3) An example of a test procedure on memory patch

Figure 6 shows an example of a test procedure on memory patch. Memory patch is a function to modify the OS kernel to be tested. The test procedure performs IPL operation of an OS three times. Thus, system programmers can confirm the influence of the three kinds of memory patch, and modify the OS kernel to operate normally.

3.5 Evaluation of OSTD/AUTO

OSTD/AUTO is evaluated from the viewpoints of operation time of IPL, OS retesting and regression testing, LCMP-configured multiple OSs testing, and operation time of tape initialize and data set copy as follows.

(1) Operation time of IPL

Table 2 shows comparison of the operation time from IPL to the Job Spooling Subsystem 3 (JSS3) start for VOS3/ES1, a large-scale OS, between interactive OSTD/AUTO operations and manual operations. The console operations include eight commands. As shown in Table 2, OSTD/AUTO during interactive operations decreases operation time to 67.2% on the average in comparison with manual operations.

(2) OS retesting and regression testing

Formerly, the periods of test preparation, test execution and program correction, which consist of OS retesting and regression testing using manual operations were in the ratio of about 2: 3:5. OSTD/AUTO, however, needs little test preparation because of its re-use of the test

```
Testing Procedure for Global Processor          Testing Procedure for Local Processor
GIPL:PROC();                                    LIPL:PROC();
        :               :                               :               :
CMD 'DEFINE CTCA C1E',TD                        COMMAND 'DEFINE CTCA C1E',TD
MATCH /ENTER COUPLE/; <-------------------------CMD 'MSG GLOBAL ENTER COUPLE',TD
CMD 'COUPLE C1E TO LOCAL C1E',TD                        :               :
CMD 'XIPL 440',TD                               CMD 'XIPL 420',TD
        :               :                               :               :
CMD 'MSG LOCAL GLOBAL READY',TD-->MATCH /GLOBAL READY/;
END GIPL;                                       CMD 'START JSS4',OS
                                                        :               :
                                                END LIPL;
```

Fig. 5 A test procedure for LCMP-configured VMs.

Table 2 Estimation of OSTD/AUTO.

No.	(a) IPL Time by Manual Operation	(b) IPL Time by OSTD/AUTO	(b)/(a) (%)	Miss Operations
1	3 min. 06 sec.	2 min. 07 sec.	68.2	no miss operations
2	3 min. 27 sec.	2 min. 13 sec.	64.3	one miss operation
3	3 min. 38 sec.	2 min. 32 sec.	69.7	in manual operation
Average	3 min. 24 sec.	2 min. 17 sec.	67.2	—

```
JOB:PROC;
  DCL &I DEC;
  &I=1;
  DO WHILE (&I<=3);
    CALL IPL;
    IF &I=1 THEN
      CALL PATCH1;
    ELSE
      IF &I=2 THEN
        CALL PATCH2;
      ELSE
        CALL PATCH3;
    COMMAND 'START TESTJOB',OS;
    COMMAND 'SLEEP',OSTD;
    COMMAND 'SYSTEM CLEAR',OSTD;
    &I=&i+1;
  END;
END JOB;
```

Fig. 6 A test procedure on memory patch.

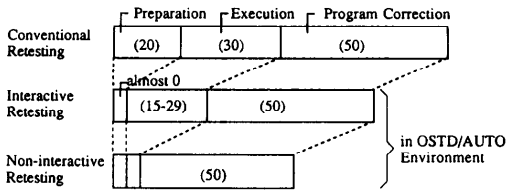

Fig. 7 Evaluation for OS retesting.

procedure. In addition, it is easy to collect test procedures for retesting and regression testing by generating them automatically from manual console operations. Thus, all regression testing can be executed in OSTD/AUTO environmet by using the collected testing procedures. Furthermore, OSTD/AUTO decreases the period of test execution for system programmers to almost 0% for non-interactive use, and to 50-95% for interactive use (depending on the frequency of the console operation). Consequently, OSTD/AUTO cuts OS test time to 50% for batch use and 65-79% for interactive use for system programmers (see **Figure 7**).

(3) LCMP-configured multiple OSs testing

Conventionally, during IPL for a loosely coupled multi-processor (LCMP) configured system, IPL had to be performed while considering the timings between the multiple processors. Multiple system programmers were occupied for the IPL operation. OSTD/AUTO, however, automates IPL for LCMP-configured multiple virtual machines; synchronization between the test procedures controlling each virtual machine enables this automation. Thus, each system programmer can perform OS testing in an LCMP environment.

(4) Operation time of tape initialize and data set copy

It needs three commands made up of 104 characters to initialize a tape and copy a data set from disk unit. The operation takes 3 minutes 3 seconds in manual operation, and 1 minute 27 seconds in OSTD/AUTO operation. Furthermore, it takes 4 minutes 9 seconds in manual operation with one miss operation. As above, OSTD/AUTO is effective for shortening the OS testing time.

4. I/O Error Simulator: IOSIM

4.1 Requirements for IOSIM

It is important for OSs to recover I/O errors for improving the reliability of computer systems. Formerly, hardware errors in the I/O devices must be supplied for the OS to test the I/O error recovery function. However, it is difficult to generate hardware errors in the I/O devices, and it takes much time to develop each hardware error generator corresponding to the I/O device. As a result, a tool to simulate all types of I/O errors independent of real hardware errors need to be developed. In addition, it is desirable to automate OS testing for I/O errors. The types of I/O errors consist of (a) errors in starting I/O operations, (b) errors during I/O processing such as unit check interrupts, and (c) repeated I/O errors. Thus, the IOSIM needs to simulate those I/O errors.

When errors in starting I/O operations or during I/O processing occur, an OS investigates

the causes of the I/O errors and tries to complete the whole I/O operations. This processing depends on the times, range and contents of errors. When repeated I/O errors occur for channel errors, an OS tries to reset the channel status to stop the repeated I/O errors and tries to make use of I/O devices under the channel. This processing depends on interval and times of the errors. Thus, those error conditions need to be specified to IOSIM by system programmers. In addition, multiple system programmers need to test error processing simultaneously.

4.2 Facilities of IOSIM

It is suitable for the VMM to simulate I/O errors because both I/O instructions issued by the OS to be tested and I/O interruptions to the OS to be tested are simulated by the VMM. Thus, IOSIM is developed in the VMM. The IOSIM can simulate errors during I/O processing, and can simulate repeated I/O errors according to the system programmers' specification. IOSIM can be used to test an OS approximately 80% of the test cases for I/O error recovery without using a particular hardware error generator. IOSIM can't specify the busy time of the I/O control units so far. IOSIM is activated by submitting a single 'IOTEST' VMM command from OSTD screen. Thus, by executing the test procedure including the 'IOTEST' command, automatic testing for I/O error recovery can be done. IOSIM has the following simulation functions (see **Table 3**):

(1) Simulation of errors in starting I/O operations

I/O error recovery in an OS performs different action according to the number of errors and range of errors. IOSIM can simulate condition codes for errors in starting I/O operations at specified times for start I/O instructions according to the specification of the 'IOTEST' command. The 'IOTEST' command can specify multiple I/O devices as devices for which errors will be simulated. When more than one I/O path are connected to one I/O device, the 'IOTEST' command can specify the error path.

(2) Simulation of errors during I/O processing

An OS decides the method to recover the I/O operation according to the error status informed by the I/O interruption. System programmers can specify information for I/O interrupts by using the 'IOTEST' command. An I/O interrupt to an OS can be delayed or deleted. IOSIM can also simulate errors for a specified channel command word (CCW) in the channel program, for specific types of channel commands and for accessing specific address areas on a disk unit by intercepting the I/O processing at specified CCW in the channel program. When a unit check interrupt is simulated, the contents of the sense byte (detailed information about the interrupt) can be also simulated to the OS according to the users' specification.

Table 3 Facilities of I/O error simulator.

No.	Facilities		Kind of simulation			
			I/O Error Instruction	I/O Interruption	Unit Check Interruption	Repeated I/O Errors
1	Specification of Target	Device	X	X	X	X
		I/O path	X	X	X	X
2	Specification of I/O	Error times	X	X	X	X
		Error interval	X	X	X	X
		Command code	—	X	—	—
		Seek address	—	X	—	—
3	Specification of Last CCW		—	X	—	—
4	Specification of Interruption Delay		—	X	—	—
5	Simulated Information	SCSW, CSW	X	X		X
		ESW, LCL	X	X	—	X
		Sense Byte	—	—	X	—

Explanation X: well suited, —: none.

(3) Simulation of repeated I/O errors

To confirm the recovery functions of the OS to be tested for repeated I/O errors, IOSIM can simulate the interrupts at specified periods using timer interrupts to VMM according to the specification of the 'IOTEST' command. According to the clear I/O instruction issued by the OS to be tested, IOSIM stops to simulate the repeated I/O errors as if the channel status were reset. Thus, an OS tries to make use of I/O devices under the channel. The above sequential actions of the testing OS for repeated I/O errors can be tested using IOSIM.

5. Conclusions

We developed the automatic OS test driver (OSTD/AUTO) and the I/O error simulator (IOSIM). OSTD/AUTO automates both OS operations and virtual machine operations by monitoring and controlling both the OS screen and the OSTD screen. OSTD/AUTO, during interactive operations, decreases operation time to about 67% on the average in comparison with manual operations. IOSIM can simulate errors during I/O processing and repeated I/O errors according to the system programmers' specification. OSTD/AUTO can be used to check and test approximately 80% of the OS test cases (such as for normal or unusual cases, conditions concerning boundaries, and combinations of these conditions) without modifying the OS to be tested. Automatic OS testing for I/O error recovery can be done by combining OSTD/AUTO and IOSIM. They can be used widely, not only to develop and test OSs, but also database systems, data communication systems, and on-line systems.

Acknowledgements OSTD/AUTO and IOSIM have been developed from a research project to improve the reliability of large-scale computer systems. They were completed thanks to the efforts of many persons in the Software Development Center, the Systems Development Laboratory, and in subsidiary companies of Hitachi, Ltd. The authors would like to particularly thank Mr. S. Takasaki, Mr. K. Totsuka, Mr. T. Hirotaka, Mr. T. Yamagishi and Mr. M. Haraguchi for fruitful technical discussions on this research.

References

1) Ramamoothy, C. V. and Ho, Siu-Bun, F.: Testing Large Software with Automated Software Evaluation Systems, *IEEE Trans. Softw. Eng.*, Vol. SE-1, No. 1, pp. 46-58 (1975).
2) Chusho, T.: Test Data Selection and Quality Estimation Based on the Concept of Essential Branches for Path Testing, *IEEE Trans. Softw. Eng.*, Vol. SE-13, No. 5, pp. 509-517 (1987).
3) Iyer, R. K. and Rossetti, D. J.: Effect on System Workload on Operating System Reliability: A Study on IBM 3081, *IEEE Trans. Softw. Eng.*, Vol. SE-11, No. 12, pp. 1438-1448 (1985).
4) Yoshizawa, Y. et al.: Test and Debugging Environment for Large Scale Operating Systems: OSTD, *COMPSAC 87*, The Computer Society of the IEEE 1987, pp. 298-305 (1987).
5) Goldberg, R. P.: Survey of Virtual Machine Research, *Computer*, Vol. 7, No. 6, pp. 34-45 (1974).
6) Meyer, R. A. and Seawright, L. H.: A Virtual Machine Time-Sharing System, *IBM Syst. J.*, Vol. 9, No. 3, pp. 199-218 (1970).
7) Kagawa S. et al.: An Automatic Operation Method in Virtual Machine System, Japanese Laid-open Patent Applications 59-718 (1984).
8) Miyazaki, Y.: A Virtual Machine with Hardware Error Generator, Japanese Laid-open Patent Applications 62-70941 (1987).

(Received November 19, 1991)
(Accepted December 3, 1992)

Shunji Tanaka (Member)

Shunji Tanaka received the B. S. and M. S. degrees in mathematics from Kyushu University in 1979 and 1982, respectively. He is a researcher in Systems Development Laboratory, Hitachi, Ltd. Since 1983, he has been engaged in research on performance improvements for virtual machine systems and reliability improvements for operating systems. His current research includes computer architectures. He is a member of the Japan Society for Software Science and Technology and the IEEE Computer Society.

Yasufumi Yoshizawa (Member)

Yasufumi Yoshizawa was born in Tokyo, Japan, in 1944. He received B. S. degree in Applied Physics from Tokyo Institute of Technology, Tokyo, Japan, in 1967 and joined Central Research Laboratory (CRL) of Hitachi, Ltd. He was a member of HITAC 5020/TSS Project in CRL. In 1973, Systems Development Laboratory (SDL), Hitachi, Ltd. has been established, and he was transferred to SDL at the same time. He researched into program behavior in virtual memory, performance evaluation of large scale time-sharing systems and on-line transaction systems and proposed several performance improvement methods. He has received an engineering doctoral dissertation based on those researches in 1981 from Tokyo Institute of Technology. He also studied for testing and debugging systems, especially operating systems. He is at present doing research work on performance improvement of workstation and server systems. He received an Award of the Excellent paper of IPSJ in 1972. And he has been an extramural lecturer of Tokyo University of Agriculture and Technology from 1980, University of Electro-Communications and Tokyo Institute of Technology from 1990. He is a chief researcher of SDL. He is an author of "Practical Operating Systems" published by Shokodo. He is a member of IEEE Computer Society.

Hidenori Umeno (Member)

Hidenori Umeno was born in Oita, Japan, in 1947. He received B. S. degree in mathematics from Kyushu University in 1970. He had been with Central Research Laboratory, Hitachi Ltd. from 1970 to 1976, and engaged in the study of productivity of compilers. Since 1976 he has been engaged in research on performance and reliability of virtual machines, logical partitions, file systems, and operating systems. His main concerns are virtual machines, logical partitions, file systems, operating systems, microprograms, and architectures. He is a member of Association for Computing Machinery and IEEE Computer Society.

Naoko Ikegaya (Member)

Naoko Ikegaya was born in Saitama, Japan. She received the B. S. degree in mathematics from Tsuda College in 1985. Since 1985 she has been with Systems Development Laboratory, Hitachi, Ltd. and has been engaged in research on control of virtual machine systems and reliability of operating systems.

Kazuyasu Miyamoto (Member)

Kazuyasu Miyamoto received B. E. degree from Yokohama National University in 1965. Since 1965, he has been with Software Development Center, Hitachi, Ltd. His current interests include all areas of software engineering, software quality assurance, and computer assisted learning. He is a member of the ISO/IEC JTC1/SC7 Committee and the IEEE Computer Society.

Nobuyoshi Sugama (Member)

Nobuyoshi Sugama was born in Ibaraki, Japan, in 1950. Since 1971, he has been with Software Development Center, Hitachi, Ltd. He had been engaged in quality assurance of software (FORTRAN compilers). Since 1975, he has been engaged in development of virtual machine system.