Published: 12/6/66

Identification

Test and Diagnostic Philosophy
Harlow E. Frick

Purpose

This section describes some of the general design considerations being taken into account on the on-line test and diagnostic system for 645 Multics installations.

Introduction

The philosophy included in this section is quite general. Test and Diagnostic philosophy as it relates to specific equipment is further discussed in Section BR.2.00.

Definitions

Test programs are defined as programs specifically written or used to determine whether a system or some part of a system is functioning properly.

Diagnostic programs are defined as programs specifically written to help field engineers diagnose hardware failures. There is no implication as to the amount or usefulness of error output information provided by test programs or diagnostic programs.

The Multics T and D System is defined as that part of the 645 software designed to aid in detection and diagnosis of hardware failures. The goal should be to implement this software so as to maximize resource availability and minimize user inconvenience in the event of hardware failures. The Multics T and D System will be specifically designed for use on operating installations in the field. The Multics T and D System will consist of four parts; Integrated tests, periodic tests, diagnostics, and utility files. A resource has an integrated test if when the resource fails to perform properly this fact will be "immediately" detected by a system or user processes and result in an indication to the operator or field engineer that something is wrong. Where practical periodic tests will be developed to test those resources which do not have complete integrated tests. When a failure is detected or suspected, the transactor will turn the resource over to the diagnostic system. If the field engineer so desires, he may run a diagnostic program on the resource in order to help him diagnose the failure. Utility files are defined as all files pointed

to by the T and D Utility directory. They contain data and procedure files required for controlling or maintaining the T and D system or for performing some useful function for field engineers or diagnostic programmers. Utility files will be further defined as the need for them arises.

Test Philosophy

The hardware should be tested with integrated tests where practical. This document describes some integrated tests for which implementation plans have already been made and recommends a few additional ones. However, recommendations should not be construed as specifications. The definition and inclusion of integrated tests, including error reporting and recovery procedures, is the responsibility of the system programmer designing the module in which the integrated test should be included. This is not to say, for example, that diagnostic system programmers will not eventually define points in the system where automatic links to diagnostics would be feasible. The role of the diagnostic system in the area of integrated tests may become more important after Multics is initially implemented.

Where integrated tests are not adequate (e.g., failures in floating point arithmetic and cards punched out of registration) an attempt will be made to provide periodic test programs. Periodic tests may be, but are not necessarily, similar to diagnostics used for corrective maintenance.

The priority and period of periodic tests are variables for each resource and are adjustable by the field engineer. The time required for a complete test depends on the component being tested. Typical time for a test might be 20 seconds for a printer test, and 60 seconds for a magnetic tape test which tests one device. (Time in this context is the total time the device is allocated to the test.) Neither integrated nor periodic tests are guaranteed to detect all types of failures. Periodic tests may miss intermittent failures. Also, no practical tests are known for some types of solid failures. However, even with these deficiencies considered, integrated and periodic tests are a very important part of the system because customers will see a more reliable system, depending on the degree to which failures are detected sooner than they would be without integrated and periodic tests.

Diagnostic Philosophy

Several diagnostic approaches are possible. Diagnostics may be on-line (i.e., run in the Multics environment), or off-line (i.e., have complete control of the system).

Diagnostics may be of a conventional organization (e.g., test from least to most complex and usually make no attempt at component board isolation), or they may depend on some advanced diagnostic technique (e.g., test in a sequence to minimize the number of tests, and attempt component board isolation. Diagnostics may be oriented to testing a single system component or to several components simultaneously.

The type of diagnostic needed for a component is affected by whether or not hardware re-configuration is required before the diagnostic is run. Therefore, let us categorize failures as to whether they will require a major reconfiguration. In this paper, a major re-configuration is defined as any hardware reconfiguration except deletion of a component (other than a system controller), or changing a peripheral to another channel, or to another GIOC. The intent of this definition is to include only hardware reconfiguration which would temporarily disrupt operation to all users while the reconfiguration was being done.

After a failure has occurred, major reconfiguration may be required in order to inhibit the failed component from causing deleterious effects on the good portion of the system. Or, it may be necessary to re-configure in order to positively determine which system component has failed. Certain hardware failures will never require reconfiguration during the process of diagnosis, repair, and operational verification. The reason these failures will never require major reconfiguration is essentially because their effect may only be observed by a limited portion of the Multics software and therefore, only this limited portion of the Multics software is affected and isolation of the failures to a specific device is usually accomplished simply by detection of a failure by this limited portion of software.

The following failures will never require major reconfiguration. These failures will henceforth be referred to as category A failures.

1. Failures in peripheral devices connected to a GIOC.
2. System clock failures.
3. EMM failures which are localized to specific EMM addresses.

The following failures are more catastrophic than those listed above. Any of these failures (henceforth be referred to as category B failures) may require major hardware re-configuration.

1. Processor failures
2. Memory failures
3. MSU failures (except as noted in category A)
4. GIOC failures

Generally Multics software, operator, or field engineer can isolate category A failures to a specific component. It is reasonable to provide diagnostics for category A failures which are oriented to diagnosing the specific component and which run in the Multics environment. This type of diagnostic will be provided for category A failures in the initial Multics T and D system and is further discussed in section 8.0. The rest of this section is related to diagnostics for category B failures.

When a category B failure occurs, we do not know how often or how reliably the system software, operator, or field engineer can immediately determine which component has failed. Therefore, upon detection of a category B failure, the system may frequently attempt to recover by a procedure similar to the following:

1. Type a message to the operator describing the failure.
2. Generate a file containing a core image as unchanged as possible from core when the failure was detected.
3. Abort the user who had the processor when the failure was detected.
4. Log each user out temporarily.
5. Type a message to the operator requesting re-configuration and Multics initialization.

The following method of re-configuration will most likely be used on the M.I.T. hardware configuration, which consists of 1 EMM, 2 processors, 2 GIOCs, and 4 system controllers. First, operation will be attempted using half of the system plus the EMM. If this fails, operation will be attempted using the other half of the system (other processor, GIOC, and 2 system controllers) plus the EMM. If this fails, the EMM is presumably the culprit. The point is that, on the M.I.T. system, excluding an EMM failure, any category B failure will result in an operational system available to normal users plus a whole system which presumably has

a failure in one of the major modules. The following three ways of diagnosing this system need to be seriously considered.

1. Use the presently existing off line T and D system.

   In many respects the off-line T and D system is not very versatile and many of the error indications are difficult to interpret. However, this system is debugged, documented, and available; and a crew of programmers is busily improving it. This system is obviously going to be used until something better comes along. On the M.I.T. configuration and on large systems in general, on-line diagnostics will most likely make this system obsolete eventually, but systems without backup modules will always need it. Therefore, continual improvement and perhaps a complete replacement of the off-line T and D system is important.

2. Develop and use off-line diagnostics using fault simulation techniques.

   A fault simulation diagnostic consists of a diagnostic program which generates a unique result for each possible failure, or small group of failures, plus a dictionary of unique results listed with the component failure, or failures, which will cause this unique result. The diagnostic is coded so as to minimize the number of instructions which must be executed in order to cause a unique result for each possible failure. More complicated instructions and patterns are usually executed first. The dictionary is generated by running the diagnostic on a logic simulator, inducing each possible failure into the logic being simulated, and compiling the unique result for each simulated failure into the dictionary.

   The effectiveness and availability of fault simulation techniques applied to the 645 are not presently known. This project is apparently progressing with no major problems. We should carefully monitor progress, define applicability to the 645 system, and define 645 hardware changes to aid in diagnostic resolution where feasible.

3. Develop and use on-line diagnostics.

   In parallel with detailed definition and implementation of the initial Multics T and D system we should launch a feasibility study to define diagnostics which should be included in the final T and D system. This study

should include an early definition and implementation of an on-line diagnostic system for either EMM, system controller, processor, or GIOC. The following comments explain why we should not attempt to include any of these diagnostics in the initial Multics T and D system.

a. Periodic tests and peripheral diagnostics will be available sooner because we can concentrate the available manpower on them. The need for periodic tests and peripheral diagnostics is immediate and and fairly well defined.

b. To embark on a concentrated on-line diagnostic effort at this time would involve either a parallel effort on fault simulation diagnostics or a huge effort on conventional diagnostics which might not be needed if fault simulation techniques prove very effective.

c. The advantage of on-line major module diagnostics is debatable and not very well defined. The debate and unclear definition will continue until we have more information. We need to know what configurations are going to be in the field. We need to know how often the software, operator or field engineer, can immediately determine which major module has failed. We need to know the effectiveness of fault simulation diagnostics. We need to know how much further degradation would be caused by using the operational part of the system to diagnose the malfunctioning part. This degradation could be considerable if conventional diagnostics were used and if a large portion of the system were suspect. For example, several system controllers, 1 processor and 1 GIOC might all be suspect.

d. Problems related to protecting the operational portion of the system from the malfunctioning part need further investigation.

Test and Diagnostic languages

Procedures in the test and diagnostic system may generally be categorized as control procedures or test procedures. Test procedures include periodic tests and diagnostics. Control procedures include:

Test and diagnostic demon
Equipment periodic test scheduler
Test and diagnostic message co-ordinator
Test and diagnostic command processors
TDL compiler

Control procedures should be coded in EPL unless there is a valid reason for not doing so. Test and diagnostic procedures should be coded in TDL if practical; otherwise they should be coded in EPL or 645 assembly language depending on which is most convenient.

TDL is a test and diagnostic language which will be available in the Multics environment. It is specifically oriented toward testing common peripheral type I O devices. This language should be machine independent and should be similar or identical to the DIAL language being developed for the Computer Equipment Department manufacturing section. Extensions to, or other versions of, this language should be considered for test and diagnosis of communications equipment, EMM, and processor.

The following introduction to DIAL has been copied from the Engineering System Spec (Timeshared Peripheral Unit Test System) which was prepared by Dick Hoehnle in May 1966.

> The acronym DIAL stands for Diagnostic Language. It is a compiled language suitable for peripheral testing. It is flexible and powerful, placing few restrictions on the most uninhibited user.
>
> The source language (user) is divided into lines and subdivided into fields. The fields are, in most cases, reduced to subfields. The fields consist of mnemonics which represent peripheral operations, data manipulation operations, error checking operations, and looping operations. A typical source language statement will appear as:
>
> ```
> 01  PWTB;TLC,1,99;PREW;PRTB;TLFC,1,4,99;END
> ```
>
> The line number will be furnished by the user. Once a few simple techniques are committed to memory, this language becomes very easy to use. First, fields are separated by (;) and represent an operation. Hence the first field (PWTB) is the operation "Write Tape Binary". Peripheral operations will always use the standard mnemonics found in the Department programming manuals, preceded by the letter P.
>
> The (PWTB;) results in all error checking being done to a standard. This would normally be ready status, terminate interrupt, and operations completed within specified time limits. The standard table will be referred to for all peripheral operations and will not be changed unless the operator specifically causes it to be.

The next field (TLC1,99) in our example is divided into subfields by commas which leads us to Definition 2 -- fields will only be subdivided by commas. This field consists of alpha-numeric and numeric data. The alpha-numeric will always precede the numeric. The (TLC) represents Transfer to Line on Count. The first character represents the class of instruction and the latter 2 describe the numeric fields that follow. First numeric field equals field and second count. What this instruction says is -- loop to line 1 if the count is less than 99. Otherwise, continue to the next field. The count will be incremented once each time this instruction is encountered.

Hence, we are going to write 100 records on mag tape. When the count is reached, we go to the next field, which is the mnemonic for Rewind followed by Read Tape Binary. The next transfer instruction functions exactly like the first one with the exception that one extra subfield has been added. It is translated as "Transfer to line 1, field 4, if the count is less than 99". This enables us to go directly to the PRTB and read the 100 records just written. END is obvious, as the end of the program.

Thus, it can be seen that with this one simple line of code, the operator can generate a program that would normally require several hundred lines of machine language code to perform the same job.

User Privacy

The field engineer occasionally has some need to perform operations which could give him access to private user files. However, the assurance of user privacy is usually more important than this need, and therefore, the test and diagnostic system must be designed in a manner which will normally protect user privacy. As the test and diagnostic system is designed and implemented, weaknesses in the system which might allow either intentional or unintentional violation of user privacy will be searched for. These weaknesses will be removed if practical, or reported if it should not be practical to remove them.

The following comments are intended to illustrate how user privacy will be protected from the diagnostic user.

1. A diagnostic user will not have access to routines which give him indirect access to restricted information. For example, he will not have access to a routine which he may use in order to read or write into all core locations. He may, however, be able to read or write into specific core locations which he has been allocated.

2. The diagnostic user will normally have access only to the following files:

    a. Files which are accessible to all ordinary users

    b. Files in the diagnostic user directory

    c. Files in his own private directory

    d. Files specified by other users as accessible to diagnostic users

3. When a system process detects incorrect data transmissions, the data transmitted will not normally be reported. For example, if an unrecoverable read error occurs on a file system device, the data read and/or the data which should have been read will not be unconditionally reported to the field engineer. However, suppose an unrecoverable read error has occurred on an EMM and the field engineer would like to re-read the "unreadable" record and/or print the data read. The system may be designed such that the field engineer always has authority to do this on records which do not contain private date, but must obtain approval of the system administrator to do so on records which may contain private data.

System protection from diagnostic users

An important feature of the on-line test and diagnostic system is protection of the operational system from blunders made by diagnostic users. The following comments are intended to illustrate the extent to which this protection will be accomplished.

1. It would be unwise to give the diagnostic user unlimited authority to use system resources, except as required in order to test and diagnose equipment problems. Therefore, the diagnostic user will log in using an account which has a moderate amount of system resources available. This is a normal account in that it allows access to procedures and data which are available to other ordinary users and it may be adjusted so as to cause special reports or automatic abortion from the system when it is overdrawn. The amount of drum or disc space may be limited and there may or may not be ability to use additional resources via standard requests to the transactor. When the diagnostic user starts a test or diagnostic routine, the account number is automatically changed to one of several diagnostic accounts. These accounts may be overdrawn to any degree without causing the program to be aborted.

2. Peripheral devices are allocated to the diagnostic system by making standard requests to the transactor. An out of service flag exists for each device which may be allocated to the test and diagnostic system. If the flag is set, the device may be allocated only to a diagnostic account. If the flag is reset, the device may be allocated only to a non-diagnostic account. The field engineer will probably have authority to execute a command which may be used to set or reset the out of service flag.

System resource de-allocation

Many situations will arise at a Multics installation which will require that a particular component be temporarily deleted from system use.

In the context of this section, a component is defined as any of the following.

1. Processor
2. GIOC
3. EMM
4. Memory controller plus all of its contained core.
5. Any device connected to a GIOC.
6. Any channel which is connected to multiple devices.

The following list is intended to illustrate the variety of situations which might require that a specific component be temporarily deleted from the system.

1. An engineering change is to be installed.
2. Preventive maintenance is to be done.
3. A particular component is suspected of causing deleterious effects on the system. Temporary disuse of this component is required to see if the problem stops occurring.
4. The component is temporarily required on another system.
5. The component malfunctions and is to be diagnosed and/or repaired off line.

6. The component malfunctions and is to be diagnosed with an on-line diagnostic. The diagnostic must have complete control of the component and therefore the Multics software must stop using it.

7. The component is to be physically moved.

There are two basic ways in which the system may stop using a component. The first way is to:

1. Log each user out.

2. Force latest copies of all segments onto EMM or disc storage.

3. Type a message indicating that the system is shut down.

4. Bootload the system and initialize in a configuration which does not include the component which is to be deleted.

When the above method is used, a similar procedure must be repeated when the component is added back into the system.

The second method can be summarized as follows. Each component has an out of service flag and an out of use flag. The field engineer will probably have authority to execute a command to set or rest the out of service flag.

The out of service flag getting set signals the system to stop using that component. The system may require several milliseconds to stop using the component (in the case of a processor) or several minutes (in the case of the EMM where files must be copied to other file system devices.) When the system has stopped using the component it sets the out of use flag and outputs a message indicating that the component is not being used. The out of service flag getting reset signals a system process to reset the out of use bit and start using the component. Implementation of this method for various components might be categorized as follows.

Should definitely be implemented for:

All peripheral devices connected to a GIOC except the disc.

Should probably be implemented for:

EMM

Processor

GIOC

Disc

Any channel connected to multiple devices

Should probably not be implemented for:

System controllers

The following examples are intended to illustrate how this mechanism might work in more detail.

Example 1 - System process or user executes a command to stop using processor 2.

1. A routine is entered which sets the processor 2 out of service flag and wakes up a routine which initiates action to stop using processor 2.

2. All processor 2 actions except interrupt processing are inhibited by placing a high priority process in the ready list which must run on processor 2 and which executes a DIS instruction. After this process is placed in the ready list, the process currently running on processor 2 is pre-empted.

3. Interrupt processing by processor 2 is inhibited by (a) masking off all interrupts from all system controllers which direct interrupts to processor 2, (b) typing instructions to the operator to switch the control port for each applicable system controller and (c) restoring the applicable system memory. controller mask registers to their original values.

4. The out of use flag for processor 2 is now set and a message is typed indicating that processor 2 is not being used.

Example 2 - Preventive maintenance is to be scheduled on a peripheral device.

This may be done in either of two ways: The first way is to schedule the peripheral device using a standard

transactor scheduling request. The planned reservation facility of the transactor may be used in order to schedule preventive maintenance on a fixed cyclic basis. Or, the request may be on an as needed basis. The disadvantage of this method is that maintenance may be required on a higher priority basis and the normal transactor requests don't override existing reservations previously made by other users. Therefore, the following mechanism will be available when it is necessary to acquire a peripheral device for preventive maintenance on a high priority basis.

A facility to set or reset an out of service flag will be provided for each peripheral device. This flag signals the system to stop using that peripheral device. Normally the user who is currently using the device will be allowed to finish his job. However, a provision must be included to interrupt operation to get some peripherals which are rather permanently assigned, like card readers, or tapes assigned to the file system. At any rate, when the process using the peripheral releases it, the out of use flag will be set and a message indicating this will be printed.

Implementation phases

The Multics T and D System will be implemented in two or more phases. The first phase, hereafter called the initial Multics T and D system, should be completely implemented by the time Multics is available for commercial use. The second phase hereafter called the final T and D system, may be implemented in several steps. The important point is that after the initial multics T and D system is implemented it must be possible to implement the final T and D system without major changes to existing Multics system modules. This means that, wherever possible, test and diagnostic considerations must be defined and implemented in the Multics system modules initially, not added in later. The initial Multics T and D system should consist of the following on-line T and D programs plus the off-line T and D system. (The presently existing off-line T and D system is described in section BR.1.)

1. A program to control user console input, compilation, and execution of a newly developed version of TDL. TDL is a symbolic Test and Diagnostic Language somewhat similar to MAD.

2. Periodic test programs for:

   Processor
   Card Punch
   Card Reader
   Printer
   Magnetic Tape
   DSU 250

3. Diagnostics for Card Punch, Card Reader, Printer, Magnetic Tape, and DSU250.

4. A limited EMM diagnostic (generally described in BR.2.00) The final T and D system should consist of the initial Multics T and D system plus diagnostics for processors, GIOCs, memories, and EMMs where feasible and useful. Additional periodic tests and diagnostics for peripheral equipment should also be included in the final T and D system.