NISTIR 7207-A

Software Write Block

Testing Support Tools Validation

Test Plan, Test Design, and Test Case Specification

March 2005

Paul E. Black
Information Technology Laboratory
National Institute of Standards And Technology
Gaithersburg, MD 20899

NIST
Technology Administration
U.S. Department of Commerce

Abstract

This NIST Internal Report consists of two parts. Part A covers the planning, design, and specification of testing and reviewing the Software write block (SWB) support tools. Part B, which is a companion document, covers the test and code review support report.

Part A gives a test plan, test design specification, and test case specification for validation of the disk drive software write block testing support tools. The test plan defines the scope, including specific items and features to be validated, the methodology or approach for validating the SWB test support tools, and some technical background. The test design specification gives requirements for validating SWB tools. These requirements yield assertions. Each assertion leads to one or more code reviews or test cases consisting of preconditions, values, and method(s) for gaining confidence that the SWB test support tools correctly assess those assertions, a test procedure and the expected results. The test case specification gives details of test and review procedures for setting up the test, performing the test, and assessing the results. Appendices include a code review checklist and source code for validation programs.

Part B reports the results of reviewing the source code of the SWB test tools and testing them against Part A of the companion NIST Internal Report entitled Software Write Block Testing Support Tools validation – Test Plan, Test Design Specification, and Test Case Specification.

The intended audience for this document should be familiar with the MS-DOS operating system, computer operation, computer hardware components such as disk drives, disk drive interfaces (e.g., IDE or SCSI) and computer forensics. A working knowledge of C and Assembly programming is not necessary for understanding but is helpful.

Keywords: Code review; computer forensic tool; software testing; software write block; testing support tools.

Contents

Table of Figures

List of Tables

Part I

Test Plan

1 Introduction

There is a critical need in the law enforcement community to ensure the reliability of computer forensics tools. The Computer Forensics Tool Testing (CFTT) program is a joint project of the National Institute of Justice, the research and development organization of the U.S. Department of Justice; the U.S. National Institute of Standards and Technology (NIST) Office of Law Enforcement Standards and Information Technology Laboratory and is supported by other organizations, including the Federal Bureau of Investigation, the Department of Defense Cyber Crime Center, and the Department of Homeland Security's Bureau of Immigration and Customs Enforcement and U.S. Secret Service. The goal of the CFTT project is to establish a methodology for testing computer forensics tools.

A *software write block* (SWB) tool is software that blocks write or modification commands from reaching a disk drive, as diagrammed in Figure 6-2. In other words, it is a program that minimizes the chance of modifying a disk drive, usually during a forensic examination of the content of the disk drive. Software Write Block Tool Specification & Test Plan [SWB TS&TP] focuses on testing this class of computer programs.

The SWBT tool package includes a program (tally13.com) that monitors interrupt 0x13 to report commands allowed or blocked, a program to deactivate tally13.com (t-off.exe), a program to record operator observations of audio or visual signals from the tool under test (sig-log.exe), and a program (test-hdl.exe) to send specific groups of commands to the software write block tool under test. The test-hdl.exe, sig-log.exe and t-off.exe programs are written in Borland C++ 4.5, and tally13.com is written in Borland Assembler. The software can be used in the MS-DOS environment to test programs such as RCMP HDL. A set of test cases for software write block tools is described in [SWB TS&TP] (see http://www.cftt.nist.gov/).

How can an analyst gain more confidence that the programs adequately test a Software Write Block tool? This document lays out a test plan and test specification to address this question.

1.1 How to read this document

This document is written assuming that the reader is familiar with the MS-DOS operating system, computer operation, computer hardware components such as disk drives, disk drive interfaces (e.g., IDE or SCSI) and computer forensics. A working knowledge of C and Assembly programming is not necessary for understanding, but is helpful.

Part I is the test plan. Sect. 2 succinctly gives this document's purpose. Sect. 3 sets forth the scope: what is and is not covered, including the items and features covered. Test methodology is discussed in Sect. 4, Sect. 5 is a bibliography of referenced documents, and Sect. 6 gives a technical background. Part II is the test specification, beginning with test requirements in Sect. 7. Sect. 8 lists test assertions. Part III is the test and review case specification, beginning with Sect. 9 which gives abstract test cases, assertion pass/fail criteria, and the detailed test procedures. Appendix A is abbreviations and acronyms. An

error checklist for code review is given in Appendix B. Finally Appendix C has the source code of the SWB test tool validation programs. The above constitutes Part A of the NIST Internal Report. A companion NIST Internal Report Part B is the test and code review summary report.

2 Purpose

This document defines requirements and tests for tools and methods used to evaluate hard disk drive *software write block* (SWB) tools used in computer forensics investigations.

3 Scope

This covers tools to test programs to prevent hard disk drives from being changed via the interrupt 0x13 interface. This does not cover hardware-based means to prevent disk drives from being changed, nor changes that might be made by directly using the BIOS interface. The primary aim is to examine if the tools correspond to their build specifications, rather than a comprehensive examination of whether the tools, used according to the test plan, adequately examine Software Write Block tools.

The expected environment is an Intel X86 or Pentium architecture PC running MS-DOS with a floppy disk and at least one hard disk drive. In particular, it applies to Windows 98. The disk drives may use either an IDE or SCSI interface. The system BIOS may be a "legacy" BIOS (does not support the interrupt 0x13 extensions) or the BIOS may support interrupt 0x13 extensions for large (more than 8GiB) disk access. This does not cover any Unix-based or other operating system.

For tools that are compiled, the test object is primarily the object or executable code. The compilation process is excluded. However the source code presumed to be the origin of the executable code constitutes another or associated test object. In plain English, the source code is reviewed for correctness, the object code is tested for behavior, and the compilation process is presumed to be faithful.

3.1 Items to Be Validated

This document uses the Computer Science sense of some terms. The verb "test" means to run a program. "Review" means to examine the source code of a program. "Validate" means to test or review to gain confidence that the program will behave as needed. Finally, "verify" means review to show that a program will satisfy its specification.[1]

There are several items to be tested or reviewed. Each item is one program.

1. Tally13

Tally13 monitors the interrupt 0x13 interface. It operates in either active or passive mode. In active mode it blocks all commands from reaching the BIOS and counts the number of

[1] Verification compares the program to what was requested: building the program right. Validation compares the program to what was needed: building the right program. For reasons ranging from mistakes to poor communication to changing needs, what was requested might not be the same as what was needed.

times each command is received for each disk. In passive mode, all commands received are passed on to the BIOS and no commands are tallied.

2. Test-hdl

Test-hdl issues interrupt 0x13 commands that are then either blocked or passed by the tool under test. The program issues commands from the six categories described in Table 8-2 of [SWB TS&TP]. Any one of the six categories can be specified or all categories can be specified.

3. T-off

T-off deactivates tally13 by switching tally13 to the passive mode.

4. Sig-log

Sig-log prompts the operator for observations of audio or visual signals produced by the tool under test.

3.2 Features to be Validated

Most features to be tested or reviewed depend on the particular item tested and are listed separately for each item. Each compiled program will be checked that it writes identifying and execution information for logging. Interactive programs must log command line information, including test case identification, operator ID, etc.

3.2.1 Tally13

1. Tally13 may be switched between passive mode and active mode.
2. In passive mode no commands are blocked.
3. In active mode no interrupt 0x13 command reaches the BIOS.
4. There is a capability to query how many of each interrupt 0x13 commands received in active mode for each disk drive.
5. An identifying value is returned via the query interface when tally13 is installed.

3.2.2 Test-hdl

If a command is issued but tally13 does not show the commands was received, it is presumed *blocked*.

These features are for use before any test commands are sent.

1. Query how many of each command was received for each disk drive. For any unexpected, log the command and disk drive.
2. If tally13 is not present, issue a message and exit.
3. Command tally13 to active mode.
4. Issue each interrupt 0x13 command in the specified category to each disk drive.
5. For each command issued, log the command code, disk drive issued to, return count, status register value and carry flag setting.

These features are for use after all test commands are sent.

6. Log the number of commands sent and the number of commands blocked for each disk drive.
7. Log that all, not all or no commands were blocked.
8. Report if any commands were counted, but not sent.

3.2.3 T-off

1. Command tally13 to passive mode.

3.2.4 Sig-log

1. Query the operator if an audio or visual signal was observed and log the operator's response.

4 Validation (Test and Review) Methodology

Gaining additional confidence that the Software Write Block test tools and plan work as designed is all the more difficult because the test tools are not grossly incorrect. That is, use of the tools to date and reports produced so far would have revealed gross errors. Therefore the test approach, as it is called in [IEEE Std 829], must concentrate on finding subtle errors or potential weaknesses.

The entire method, the plan and the tools together, could be validated. That is, the plan and the tools would be checked against a general set of requirements for SWB test tools. This comprehensive approach would take more time than it is worth. Instead each tool has its own, separate, explicit requirements, against which it is validated. Exhaustive testing of the tools is also possible, but not justified given the time needed.

This document uses the term "validate", instead of test, for several reasons. First, it may be confusing to talk about, for instance, test case test cases, that is, cases to test other test cases. Also, one of the methods is manual code inspection or review, not executable tests. Most importantly, standard documents say that institutions "shall validate non-standard methods" [NIST HB 150] Sect. 5.4.5.2.

4.1 Test and Review Approach

One possible approach is to essentially repeat what has been done before, by creating independent versions of the tools and plan, test the same SWB tools again, and compare the results. This would lack crucial independence: the examiner would be writing similar tools for similar purposes and running similar tests on similar machines. We would expect the examiner to be prone to make "equivalent logical errors" or different logical errors with "statistically correlated failures" [BKL N-Version].

Instead the approach includes several different methods to validate the SWB test tools. Code is reviewed or inspected for possible anomalous behavior [dACBP Prac] and to

determine if it meets its requirements. Compiled versions of the programs are executed in very simple, controlled tests to independently check behavior. Information from code reviews and information from test runs are combined to support assertions.

It should be noted that no methodology could hope to be complete for programs of more than minimally complexity. An extreme example would be an SWB tool with a so-called "Trojan horse", i.e. a piece of code to perform a very particular unexpected function, that *does* change a protected disk if the computer's date is, say, Sunday, 6 April 2014. While inspecting the code might catch such an obvious problem, other anomalous behavior can be arbitrarily subtle. Therefore testing and review continues until a sufficient level of confidence is warranted or the test budget is exhausted.

These methods are likely to find simple errors, but will they find complex, subtle, or non-localized errors? Researchers have found support [DLP Hints] [Offutt] [KWG Interact] for the "fault-coupling hypothesis": tests for simple faults are likely to find complex faults, too. Thus, there is support that checking for simple errors suffices.

4.2 Validation Phases

The features to be tested and reviewed are divided into several groups so the tools can be validated in phases. As pointed out above, duplicating work already done to develop SWB test tools increases the cost of tests and would only slightly increase independence. However the nature of interrupt 0x13 service handling is such that validating the SWB test tools requires much the same functionality as the SWB test tools themselves!

Clearly the examiner needs to test that the driver sends the intended interrupt 0x13 commands. Other than inspecting the code, one simple strategy would be to run the driver then examine the disk drive for evidence of the commands. The limitation is that only effects of certain command parameters are measured: some status commands are defined to be handled by the BIOS without sending anything to the disk. A second limitation is that some commands should only be executed in a factory setting. Some commands in the configuration or control category may have subtle and unexpected results. The proper parameter values are propriety, and the user is cautioned that improper parameters may render a disk drive unusable. What is a good way to test that the driver sends such potentially destructive commands without the risk of damaging disk drives? Intercept interrupt 0x13 commands and report how many and which kind were sent. But this is essentially the function of tally13.

The tests are broken into independent pieces as much as practical so everything does not have to be retested if one tool changes. Once an item is validated, it can be used in subsequent phases to validate other items.

5 References

[BKL N-Version] Susan S. Brilliant, John C. Knight and Nancy G. Leveson, “Analysis of Faults in an *N*-Version Software Experiment”, IEEE Transactions on Software Engineering, 16(2):363-377, February 1990.

[dACBP Prac] Jorge Rady de Almeida Jr., João Batista Camargo Jr., Bruno Abrantes Basseto, and Sérgio Miranda Paz, “Best Practices in Code Inspection for Safety-Critical Software”, IEEE Software, 20(3):56-63, May/June 2003.

[DLP Hints] Richard A. De Millo, Richard J. Lipton and Frederick G. Sayward, “Hints on Test Data Selection: Help for the Practicing Programmer”, IEEE Computer, 11(4):34-41, April 1978.

[IEEE Std 829] IEEE Std 829-1998, Standard for Software Test Documentation, Institute of Electronic and Electrical Engineers, 1998.

[KWG Interact] D. Richard Kuhn, Dolores R. Wallace, and Albert M. Gallo, Jr., “Software Fault Interactions and Implications for Software Testing”, IEEE Transactions on Software Engineering, 30(6):418-421, June 2004.

[NIST HB 150] NIST Handbook 150, 2001 ed. “National Voluntary Laboratory Accreditation Program; Procedures and General Requirements.”

[Offutt] A. Jefferson Offutt, "Investigations of the Software Testing Coupling Effect", ACM Trans. on Software Engineering Methodology, 1(1):3-18, January 1992.

[SWB TS&TP] Software Write Block Tool Specification & Test Plan, Version 3.0, September 1, 2003. http://www.cftt.nist.gov/documents/SWB-STP-V3_1a.pdf accessed 24 February 2004.

[SWBTT] SWBT 1.0: Software Write Block Testing Tools, Requirements, Design Notes and User Manual, Version 1.0, October 2003.

6 Technical Background

6.1 Software Write Block Tools

The basic function of a software write block tool is to not allow a protected disk drive to be modified while still allowing any access that does not modify the disk drive. The critical requirements are the following:

- The tool shall not allow a protected disk drive to be changed.
- The tool shall allow any information to be obtained from or about any disk drive.
- The tool shall allow any operations to a disk drive that is not protected.

The next subsection reviews how disk drives are attached to a computer and how programs running on the computer access them. Sections 6.3 and 6.4 explain how this changes when using a software write block tool and when testing an SWB tool. Sections 6.5 and 6.6 define terminology related to software write block tools.

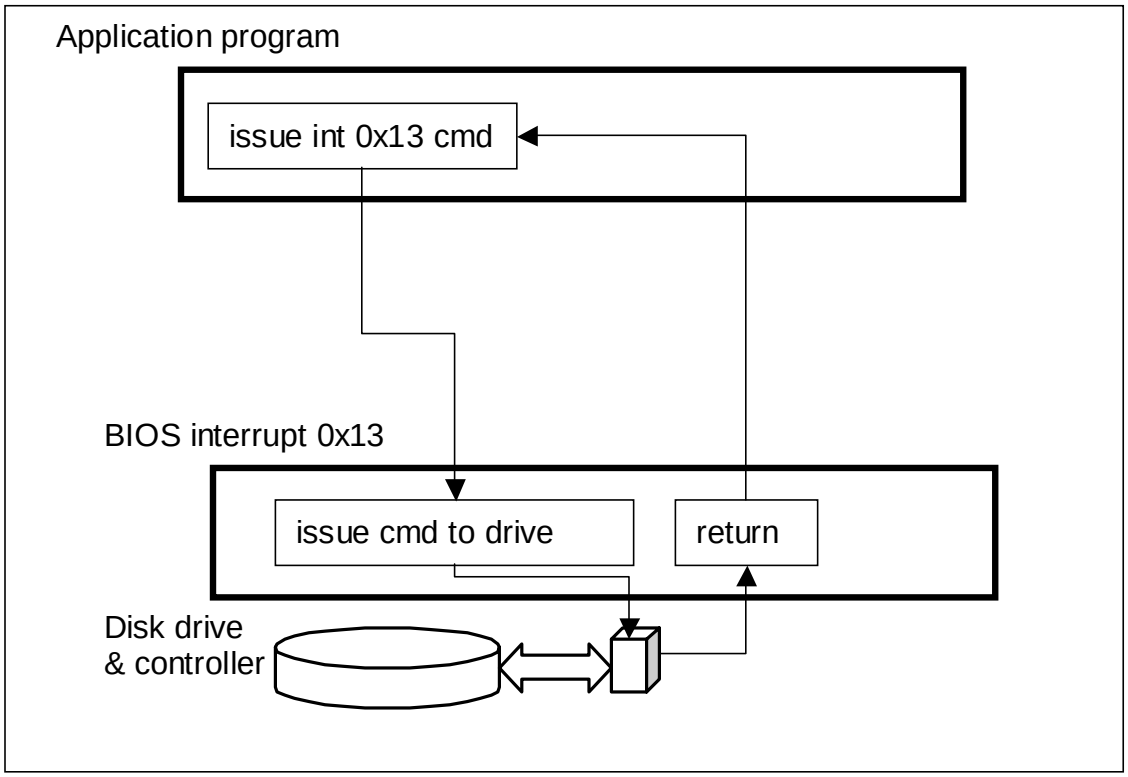
6.2 Disk Drive Attachment and Access

Before a disk drive can be used it must have some electronically connection to a computer. A disk drive is attached to a computer by one of several available physical interfaces. A disk drive is usually connected by a cable to a drive controller located either on the motherboard or on a separate adapter card. The most common interfaces for communicating with the disk drive through the drive controller are the AT Attachment (ATA) or Integrated Drive Electronics (IDE) interface. These include variants such as Enhanced IDE (EIDE) or ATA-2, ATA-3, etc. Other physical interfaces include, but are not limited to Small Computer System Interface (SCSI), IEEE 1394 (also known as FireWire or i.Link) and Universal Serial Bus (USB).

All access to a disk drive is accomplished by commands sent from a computer to a disk drive through the drive controller. However, since the low level programming required for direct access through the drive controller is difficult and tedious, operating systems usually provide other access interfaces. For example, programs running in the MS-DOS environment can, in addition to direct access via the drive controller, use two other interfaces: MS-DOS service interface (interrupt 0x21) or BIOS service interface (interrupt 0x13). The MS-DOS service operates at the logical level of files and records (the “file system”) while the BIOS service operates at the physical drive sector level. More sophisticated operating systems, for example Windows XP or a UNIX variant (e.g., Linux), may disallow low level interface (through the BIOS or the controller) and only allow user programs access to a disk drive through a device driver, a component of the operating system that manages all access to a device.

Using the interrupt 0x13 interface for disk drive access is illustrated in Figure 6-1. An application program issues an interrupt 0x13 command. The interrupt transfers control to the BIOS interrupt 0x13 routine. The BIOS routine issues commands directly to the disk drive controller. The device does the requested operation and returns the result to the BIOS which then returns it to the application program.

Figure 6-1 Disk drive Access Through the 0x13 BIOS Interface

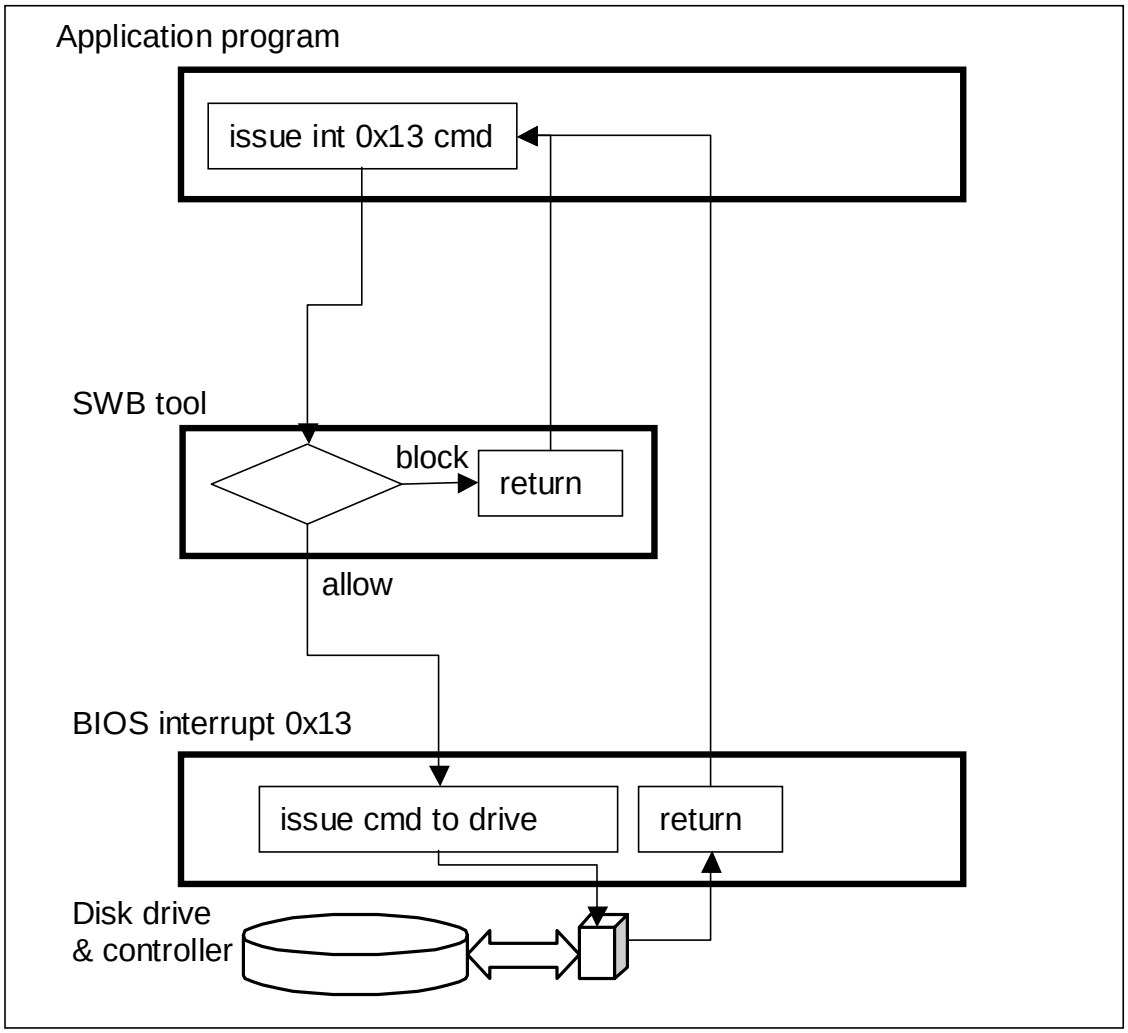


6.3 Software Write Block Tool Operation

Using a SWB tool changes the normal operation of the interrupt 0x13 interface. Figure 6-2 illustrates a SWB tool in operation. When first executed, the SWB tool saves the current interrupt 0x13 routine entry address (labeled BIOS interrupt 0x13) and installs a new interrupt 0x13 routine (labeled SWB tool). Subsequently interrupt 0x13 commands are handled in the following steps.

1. The application program initiates a disk drive I/O operation by invoking interrupt 0x13 that goes to the replacement routine installed by the SWB tool.
2. The SWB tool determines if the requested command should be blocked or if the command should be allowed.
3. If the SWB tool blocks the command, the tool returns to the application program without passing any command to the BIOS interrupt 0x13 routine. Depending on tool configuration, either **success** or **error** is returned for the command status code.
4. If the command is allowed (not blocked), the command is passed to the BIOS and the BIOS I/O routine issues required I/O commands to the drive controller so that the desired I/O operation occurs on the disk drive.
5. Results are returned to the application program.

Figure 6-2 SWB Tool Operation



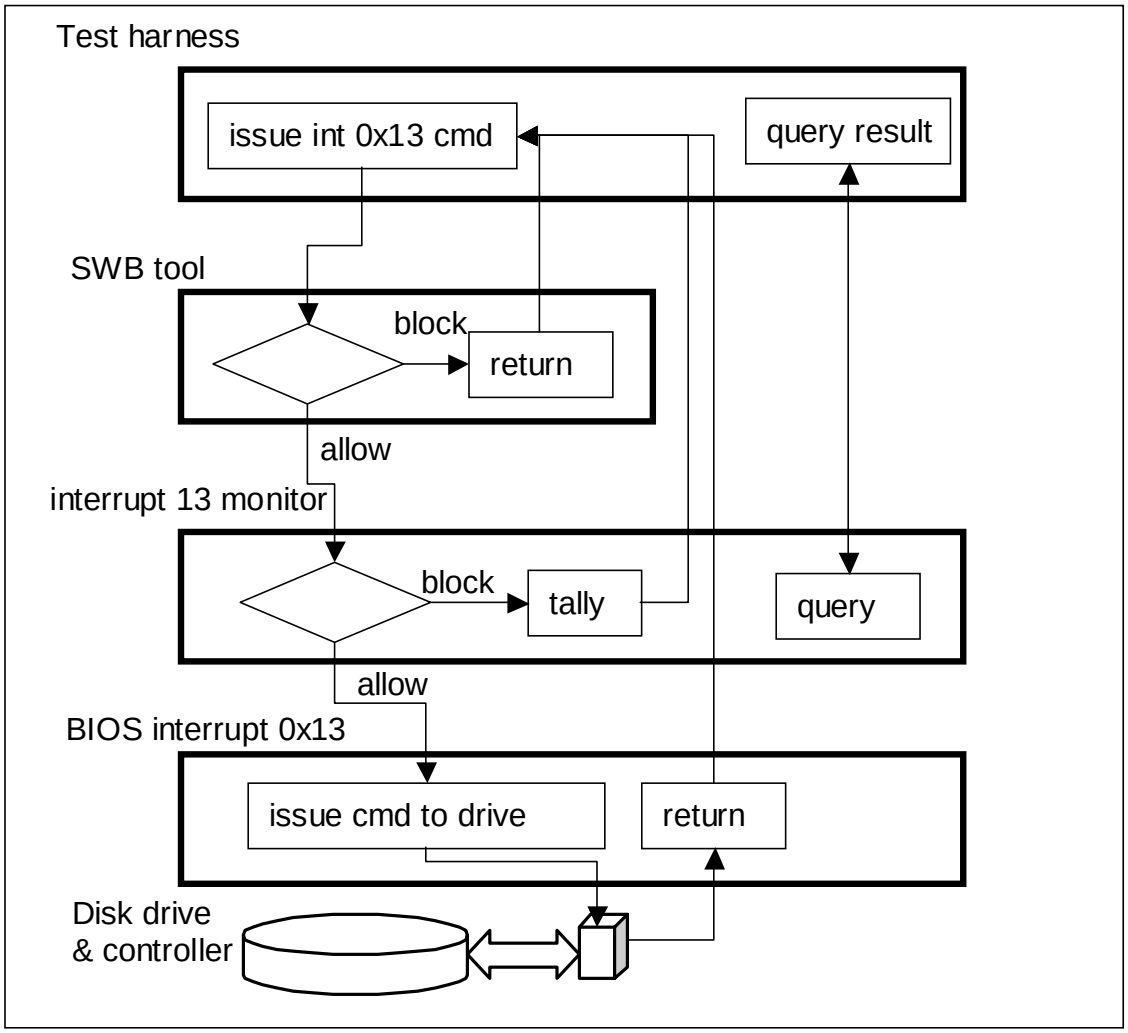
6.4 SWB Tool Test Methodology

This section describes the methodology that has been developed to test interrupt 0x13 based SWB tools. The normal interrupt 0x13 BIOS routine is replaced with a software monitor, called tally13, that counts the number of times each I/O function is called for each disk drive. When active tally13 blocks all commands from reaching the BIOS so that any command can be safely issued to a SWB tool. Tally13 has a secondary interface to allow a test harness to query tally13 to determine the command functions blocked or allowed by the SWB tool.

Tally13 operates in two modes: *allow command* or *block-and-tally*. In the *allow command*, or passive, mode all commands are passed to the disk drive. The *allow command* mode permits the SWB tool to initialize during installation. After the SWB tool is installed, tally13 is switched to *block-and-tally*, or active, mode. In this mode, tally13 blocks all commands passed by the SWB tool and a count by command and also by disk is kept of all commands seen by tally13.

Figure 6-3 illustrates the command flow in the configuration ultimately examined in a test case. First, tally13 is installed to replace the usual interrupt 0x13 processing. After the interrupt 0x13 monitor and the SWB tool are installed, the test harness is executed. The test harness issues every command for a given category. For example, a test of the read category issues each command defined for the category: 0x02 (read), 0x10 (read long) and 0x42 (extended read). As each command is issued, the SWB tool receives the command and either blocks (return with no action) or allows the command (passes it to the interrupt 0x13 monitor). If the command gets to tally13, a separate tally for each received command and disk is incremented and tally13 returns to the caller (the test harness). After control returns to the test harness, the test harness queries the interrupt 0x13 monitor to get a count of the number of times the issued command has been intercepted by tally13. If the count is zero, the SWB tool blocked the command. A non-zero count indicates that the command was not blocked.

Figure 6-3 Test Harness and Interrupt Monitor Operation



6.5 Technical Terminology

The following terms are defined for convenience in specifying the tool requirements.

- **Covered interface**: a disk drive access interface that is controlled by the SWB tool under test.
- **Covered drive**: a disk drive attached to a covered interface.
- **Protected drive**: a disk drive designated for protection from modification when accessed by a covered interface.
- **Unprotected drive**: a disk drive that is not protected from modification through a specified access interface.
- **(Send a) command**: when an application issues a command to a disk drive using the interrupt 0x13 interface.
- **Block (a command)**: when a SWB tool receives a command sent by an application but neither the issued command nor any other command is sent to the disk drive.
- **Blocked command to a drive**: a command issued by an application program that is intercepted by a SWB tool such that neither the issued command nor any other command is sent to the disk drive.
- **Return value**: a value returned to the application from the interrupt 0x13 service routine indicating success or failure.

6.6 Types of Commands

Commands for the interrupt 0x13 interface can be partitioned into two general categories: those that may change a disk and those that definitely will not change a disk. Each category has several different subcategories.

Commands that May Change a Disk

- **Write**: commands that transfer data from the computer memory to the disk drive.
- **Configuration**: commands that change how the disk drive is presented to the host computer. This type of command often destroys data on the disk drive or makes data inaccessible.
- **Miscellaneous**: commands that do not fit into the other categories.

Commands that Do Not Change a Disk

- **Read**: commands that transfer data from the disk drive to the computer memory.
- **Control**: commands that request the disk drive to do a nondestructive operation, for example: reset or seek.
- **Information**: commands that return information about the disk drive.

Part II

Test and Review Design Specification

7 Tool Requirements

This section presents requirements for mandatory features for the software write block (SWB) tools. There are no optional features: the requirements parallel the build requirements for SWB tools in [SWBTT], Sect. 2. The verb "logs" means writes to a log file.

7.1 Requirements for Mandatory Features

The mandatory features are divided into common or shared requirements and one subsection for each test item (see Sect. 3.1).

STV-RM-01. When executed, each program must write the following information to stdout or a log file:

1. Program name.
2. For each source file, source file name, version number, and date and time the source file was created.
3. System date and time program execution begins.
4. A compiled program must also write date and time the program was compiled.

STV-RM-02. When executed, interactive programs must write the following information to stdout or a log file:

1. Command line, including any command line options.
2. Test case ID.
3. Command category to be tested, if applicable.
4. ID (initials or name) of the operator.
5. Name of the computer on which the program is executed.

7.1.1 Tally13

Since SWB testing depends little[2] on what tally13 does with interrupt 0x13 commands after it receives them, there are only simple requirements on allowing and blocking commands.

STV-RM-03. Tally13 shall start running in passive mode.

STV-RM-04. Tally13 may be switched between passive mode and active mode. (see STV-RM-10 and STV-RM-18)

STV-RM-05. Tally13 stays in its mode, either active or passive, until switched.

STV-RM-06. Tally13 allows how many of each interrupt 0x13 command is received in active mode for each disk drive to be retrieved. (see STV-RM-11, STV-RM-14, STV-RM-15, and STV-RM-16)

STV-RM-07. An identifying value is returned via the query interface when tally13 is present. (see STV-RM-09 and STV-RM-17)

[2] When SWB tools start, they must read the numbers of drive and other related information. Therefore tally13 must allow commands initially.

7.1.2 Test-hdl

Test-hdl is an interactive program, so STV-RM-02 applies. The following also apply:

STV-RM-08. When executed, test-hdl must write the following information to stdout or a log file:
1. The number of installed disk drives.
2. The external labels of each disk drive.

The following requirements are for use before any test commands are sent.

STV-RM-09. If tally13 is not present, test-hdl issues a message and exits. (see STV-RM-07)
STV-RM-10. Test-hdl commands tally13 to active mode. (see STV-RM-04)
STV-RM-11. If the number of each command received for each disk drive is not zero, test-hdl logs the command and disk drive. (see STV-RM-06)

These are for sending test commands.

STV-RM-12. Test-hdl sends each interrupt 0x13 command in the specified category to each disk drive.
STV-RM-13. For each command sent, test-hdl logs
1. The command code,
2. Disk drive issued to,
3. Return count,
4. Status register value, and
5. Carry flag setting.

These are for use after all test commands are sent. (see STV-RM-06)

STV-RM-14. Test-hdl logs how many of each command is sent and how many of each command is blocked for each disk drive.
STV-RM-15. Test-hdl logs that all, not all or no commands were blocked for each disk drive.
STV-RM-16. Test-hdl logs if any commands were received, but not sent.

Commands would only be counted as received but not sent if there is a failure in one of the tools. The failure may be that the tally13 counts or reports incorrectly, there is a bug in test-hdl (sending, but claiming not to have sent), the SWB spontaneously sends interrupt 0x13 commands, etc.

7.1.3 T-off

T-off is an interactive program, so STV-RM-02 applies. The following also apply:

STV-RM-17. If tally13 is not present, t-off issues a message and exits. (see STV-RM-07)

STV-RM-18. T-off commands tally13 to passive mode. (see STV-RM-04)

7.1.4 Sig-log

Sig-log is an interactive program, so STV-RM-02 applies. The following also apply:

STV-RM-19. Sig-log asks the operator if an audio or visual signal was observed and logs the response.

7.2 Requirements for Optional Features

There are no optional features.

8 Assertions

Each assertion specifies a condition or set of conditions and the expected result. Following each assertion are criteria to determine if the assertion is supported. The particular step in the test case or code review appears in brackets. [CCC-NN (x)] means step x of test case or code review CCC-NN.

In assertions, the verb "logs" means writes to a log file.

8.1 Assertions for Mandatory Features

This section lists assertions the tools must meet. There is one subsection for each tool.

8.1.1 Tally13

STV-AM-01. When executed, tally13 writes the following information to stdout: program name, file names, version numbers and date and time of creation of each source file, system date and time program execution begins, and date and time compiled.

Criteria: passes code review [T13-01 (b)], information gathered [T13-01 (a)] matches that observed when program runs [T13-02 (b)], and execution time [T13-02 (a)] matches that observed when program runs [T13-02 (b)].

STV-AM-02. Tally13 shall start running in passive mode.

Criteria: passes code review [T13-01 (b)] and disk is readable [T13-02 (c)].

STV-AM-03. Tally13 may be switched from passive to active mode.

Criteria: passes code review [T13-01 (b)] and disk is no longer readable [T13-02 (e)].

STV-AM-04. Tally13 may be switched from active to passive mode.

Criteria: passes code review [T13-01 (b)] and disk is again readable [T13-02 (g)].

STV-AM-05. Tally13 stays in passive mode until switched.

Criteria: passes code review [T13-01 (b)] and disk stays readable [T13-02 (h)].

STV-AM-06. Tally13 stays in active mode until switched.
Criteria: passes code review [T13-01 (b)] and disk stays unreadable [T13-02 (f)].

STV-AM-07. Tally13 allows how many of each interrupt 0x13 command is received in active mode for each disk drive to be retrieved.
Criteria: passes code review [T13-01 (b)] and counts match commands sent [T13-03 (e)].

STV-AM-08. An identifying value is returned via the query interface when tally13 is present.
Criteria: passes code review [T13-01 (b)] and the value is returned [T13-02 (d)].

8.1.2 Test-hdl

STV-AM-09. When executed, test-hdl logs the following information: program name, file names, version numbers and date and time of creation of each source file, system date and time program execution begins, and date and time compiled.
Criteria: passes code review [THDL-01 (b)], information gathered [THDL-01 (a) andTHDL-02 (a)] matches that observed when program runs [THDL-02 (b)], and execution time [THDL-02 (a)] matches that observed when program runs [THDL-02 (b)].

STV-AM-10. When executed, test-hdl logs the following information: command line, including any command line options, test case ID, command category to be tested, operator ID (initials or name), and name of the computer on which the program is executed.
Criteria: passes code review [THDL-01 (b)] and logs command line information [THDL-04 (a)].

STV-AM-11. When executed, test-hdl writes the following information to a log file: the number of installed disk drives and the external labels of each disk drive.
Criteria: passes code review [THDL-01 (b)] and logs disk information [THDL-04 (a)].

The following requirements are for use before any test commands are sent.

STV-AM-12. If tally13 is not running, test-hdl issues a message and exits.
Criteria: passes code review [THDL-01 (b)], issues a message, and exits [THDL-02 (b)].

STV-AM-13. Test-hdl commands tally13 to active mode.
Criteria: passes code review [THDL-01 (b)] and disk is not readable [THDL-04 (b)].

STV-AM-14. If the number of each command received for each disk drive is not zero, test-hdl logs the command and disk drive.
Criteria: passes code review [THDL-01 (b)] and test-hdl exits [THDL-03 (b)].

These are for sending test commands.

STV-AM-15. Test-hdl sends each interrupt 0x13 command in the specified category to each disk drive.
Criteria: passes code review [THDL-01 (b)] and vtreport counts the expected number of commands [THDL-04 (c) and THDL-05 (b)].

STV-AM-16. For each command sent, test-hdl logs the command code, disk drive issued to, return count, status register value, and carry flag setting.
Criteria: passes code review [THDL-01 (b)] and logs command information [THDL-04 (a) and THDL-05 (a)].

These are for use after all test commands are sent.

STV-AM-17. Test-hdl logs how many of each command is sent and how many of each command is blocked for each disk drive.
Criteria: passes code review [THDL-01 (b)] and logs command and blocking information [THDL-04 (a), THDL-05 (a), and THDL-06 (a)].

STV-AM-18. Test-hdl logs that all, not all or no commands were blocked for each disk drive.
Criteria: passes code review [THDL-01 (b)] and logs command blocking information [THDL-04 (a), THDL-05 (a), and THDL-06 (a)].

STV-AM-19. Test-hdl logs if any commands were received, but not sent.
Criteria: passes code review [THDL-01 (b)] and logs that some unsent commands were received [THDL-05 (a)].

8.1.3 T-off

STV-AM-20. When executed, t-off logs the following information: program name, file names, version numbers and date and time of creation of each source file, system date and time program execution begins, and date and time compiled.
Criteria: passes code review [TOFF-01 (b)], information gathered [TOFF-01 (a) andTOFF-02 (a)] matches that observed when program runs [TOFF-02 (a)], and execution time [TOFF-02 (a)] matches that observed when program runs [TOFF-02 (a)].

STV-AM-21. When executed, t-off logs the following information: command line, including any command line options, test case ID, operator ID (initials or name), name of the computer on which the program is executed.
Criteria: passes code review [TOFF-01 (b)] and logs command line information [TOFF-02 (a)].

STV-AM-22. If tally13 is not present, t-off issues a message and exits.
Criteria: passes code review [TOFF-01 (b)] and issues a message and exits [TOFF-02 (a)].

STV-AM-23. T-off commands tally13 to passive mode.
Criteria: passes code review [TOFF-01 (b)] and disk is again readable [TOFF-03 (b)].

8.1.4 Sig-log

STV-AM-24. When executed, sig-log logs the following information: program name, file names, version numbers and date and time of creation of each source file, system date and time program execution begins, and date and time compiled.
Criteria: passes code review [SIGL-01 (b)], information gathered [SIGL-01 (a) and (b)] matches that observed when program runs [SIGL-02 (b)], and execution time [SIGL-02 (a)] matches that observed when program runs [SIGL-02 (b)].

STV-AM-25. When executed, sig-log logs the following information: command line, including any command line options, test case ID, operator ID (initials or name), name of the computer on which the program is executed.
Criteria: passes code review [SIGL-01 (b)] and logs command line information [SIGL-02 (b)].

STV-AM-26. Sig-log asks the operator if an audio or visual signal was observed and logs a positive response.
Criteria: passes code review [SIGL-01 (b)] and logs a signal [SIGL-02 (b)].

STV-AM-27. Sig-log asks the operator if an audio or visual signal was observed and logs a negative response.
Criteria: passes code review [SIGL-01 (b)] and logs no signal [SIGL-03 (a)].

8.2 Assertions for Optional Features

There are no optional features.

Part III

Test and Review Case Specification

9 Test and Review Procedures

9.1 Environmental Setup

This section covers the hardware and software needed to validate the SWB test tools.

On Windows 98 the "dir" command seems to cache the latest result. Therefore testing must do some other "dir", e.g., `dir a:`, to "flush" the cache before "dir" can be used to determine if commands are going to the disk drive.

9.1.1 Hardware for testing

Install one or more hard disk drives in the computer so one is drive C.

9.1.2 Software for Phase 1: tally13

The programs listed in Table 9-1 are required for testing in phase 1.

Table 9-1 Software Required for Phase 1 Testing

Program	Description
tally13	The interrupt 0x13 monitor program. Log compile and run information. Block all interrupt 0x13 command functions, count the number of times each function is requested for each disk drive, and provide an interface for retrieving those counts for each disk drive.
vtpass	Command tally13 to passive mode. (A validation version of t-off.)
vtact	Command tally13 to active mode. Report if tally13 returns an identifying value.
vtcmdgrp	Send every command with bit N on to the specified disk drive. For N=0, send 0x1, 0x3, 0x5, etc. For N=1, send 0x2, 0x3, 0x6, 0x7, etc.
vtreport	Query tally13 for number and type of commands blocked and report them.

9.1.3 Software for Phase 2: test-hdl

The programs listed in Table 9-2 are required for testing in phase 2.

Table 9-2 Software Required for Phase 2 Testing

Program	Description
tally13	The interrupt 0x13 monitor program.
test-hdl	The interrupt 0x13 driver and reporter program. Log compile and run information. Issue (request) all interrupt 0x13 commands for a specified command category. Query tally13 to determine if the

Program	Description
	function was allowed (received) or blocked (not received).
vtblksom	Change all interrupt 0x13 commands with a function of 0x80 (an arbitrary function) to function 0x81 and pass. Block all other interrupt 0x13 commands greater than 0x40 (an arbitrary cut off).
vtblock	Block all interrupt 0x13 commands.

9.1.4 Software for Phase 3: t-off and sig-log

The programs listed in Table 9-3 are required for testing in phase 3.

Table 9-3 Software Required for Phase 3 Testing

Program	Description
tally13	The interrupt 0x13 monitor program.
vtact	Command tally13 to active mode.
t-off	Log compile and run information. Exit if tally13 is not running. Command tally13 to passive mode.
sig-log	Log compile and run information. Ask the operator if a signal was observed, and log the response.

9.2 Code Review and Test Case Specifications

Phase 1 validates tally13.

9.2.1 Code Review T13-01

9.2.1.1 Purpose

Collect information about the source code to check that the right information is reported [STV-AM-01]. Review the source code to find possible errors and to check that it meets all assertions [STV-AM-01 through STV-AM-08].

9.2.1.2 Setup

(a) Get the source code for tally13.
(b) For each reviewer, get a copy of the source code, a copy of the error checklist, a copy of the assertions, and a copy of other conditions to check.

9.2.1.3 Dependencies

None.

9.2.1.4 Procedure

(a) Find file names, version numbers, and date and time of creation of source files.
(b) Review the code. In addition to the error checklist, check that it satisfies STV-AM-01 through STV-AM-08, check the conditions for which counts overflow and what

happens when they overflow, and check what happens if there are more than five disk drives.

9.2.1.5 Expected results

(a) The file names, version numbers, and date and time of creation of source files.
(b) Possible violations of the error checklist, assertions STV-AM-01 through STV-AM-08, count overflow handling, handling in case of more than five disk drives, and commanding to present mode.

9.2.2 Test Case T13-02

9.2.2.1 Purpose

Check that tally13 reports file information and date and time run [STV-AM-01], starts in passive mode [STV-AM-02], can be switched between active and passive mode [STV-AM-03 and STV-AM-04], stays in mode until switched [STV-AM-05 and STV-AM-06], and returns an identifying value [STV-AM-08].

9.2.2.2 Setup

(a) Standard computer setup.
(b) Run `dir c:`. Note directory information. (It should be blocked when tally13 is active.)

9.2.2.3 Dependencies

None.

9.2.2.4 Procedure

(a) Note system date and time. [STV-AM-01 (report execution time.)]
(b) Run tally13. Observe stdout. [STV-AM-01]
(c) Run `dir a:` (to flush cache). Run `dir c:`. Observe result. [STV-AM-02 (starts passive)]
(d) Run vtact. Observe result. [STV-AM-08 (returns identifying value)]
(e) Run `dir a:` (to flush cache). Run `dir c:`. Observe result. [STV-AM-03 (switch to active mode)]
(f) Run vtact. Run `dir a:` (to flush cache). Run `dir c:`. Observe result. [STV-AM-06 (stays active)]
(g) Run vtpass. Run `dir a:` (to flush cache). Run `dir c:`. Observe result. [STV-AM-04 (switch to passive mode)]
(h) Run vtpass. Run `dir a:` (to flush cache). Run `dir c:`. Observe result. [STV-AM-05 (stays passive)]

9.2.2.5 Expected results

(a) n/a
(b) Tally13 reports program name, file information, and system time.
(c) Dir command returns expected result.
(d) Vtact reports an identifying value returned.

(e) Results are consistent with commands to disk being blocked.
(f) Results are consistent with commands to disk being blocked.
(g) Dir command returns expected result.
(h) Dir command returns expected result.

9.2.3 Test Case T13-03

9.2.3.1 Purpose

Check that tally13 counts commands received in active mode [STV-AM-07].

9.2.3.2 Setup

(a) Standard computer setup.

9.2.3.3 Dependencies

None.

9.2.3.4 Procedure

(a) Run tally13.
(b) Run `dir a:` (to flush cache). Run `dir c:`.
(c) Run vtreport and observe stdout.
(d) Run vtact (make tally13 active).
(e) Run vtcmdgrp with the parameters in Table 9-4. Each time run vtreport and append to the log file. [STV-AM-07 (commands counted)]

9.2.3.5 Expected results

(a) n/a
(b) n/a
(c) Vtreport should show that tally13 did not count any commands. (A `dir c:` sends 0x02 commands to the first hard disk "80".)
(d) n/a
(e) Vtreport responds with the results in Table 9-4.

Table 9-4 Vtcmdgrp Parameters and Expected Vtreport Results.

	vtcmdgrp parameters		
Run #	Bit	Drive	Increased Counts For Commands
1	0	1	*1, *3, *5, *7, *9, *b, *d, and *f
2	1	2	*2, *3, *6, *7, *a, *b, *e, and *f
3	2	3	*4, *5, *6, *7, *c, *d, *e, and *f
4	3	4	*8, *9, *a, *b, *c, *d, *e, and *f
5	4	2	1*, 3*, 5*, 7*, 9*, b*, d*, and f*
6	5	3	2*, 3*, 6*, 7*, a*, b*, e*, and f*
7	6	1	4*, 5*, 6*, 7*, c*, d*, e*, and f*

Phase 2 tests test-hdl. Since tally13 will have been validated, it can be used to validate test-hdl.

9.2.4 Code Review THDL-01

9.2.4.1 Purpose

Collect information about the source code to check that the right information is reported [STV-AM-09]. Review the source code to find possible errors and to check that it meets all assertions [STV-AM-09 through STV-AM-19].

9.2.4.2 Setup

(a) Get the source code for test-hdl.
(b) For each reviewer, get a copy of the source code, a copy of the error checklist, and a copy of the assertions.

9.2.4.3 Dependencies

None.

9.2.4.4 Procedure

(a) Find file names, version numbers, and date and time of creation of source files.
(b) Review the code. In addition to the error checklist, check that it satisfies STV-AM-09 through STV-AM-19.

9.2.4.5 Expected results

(a) The file names, version numbers, and date and time of creation of source files.
(b) Possible violations of the error checklist and assertions STV-AM-09 through STV-AM-19.

9.2.5 Test Case THDL-02

9.2.5.1 Purpose

Check that test-hdl reports file information and date and time run [STV-AM-09]. Check that it issues a message and exits if tally13 is not running [STV-AM-12].

9.2.5.2 Setup

(a) Standard computer setup. Note: Tally13 NOT installed.

9.2.5.3 Dependencies

None.

9.2.5.4 Procedure

(a) Note system date and time. [STV-AM-09 (report execution time)]
(b) Run test-hdl THDL-02 USR HOST a HDD1 HDD2 … Observe stdout. [STV-AM-09 and STV-AM-12 (message and exit if no tally13)]

9.2.5.5 Expected results

(a) n/a
(b) Test-hdl reports program name, file information, and system time. It also issues a message that tally13 is not running and exits.

9.2.6 Test Case THDL-03

9.2.6.1 Purpose

Check that test-hdl exits if initial counts are not zero [STV-AM-14].

9.2.6.2 Setup

(a) Standard computer setup.

9.2.6.3 Dependency

Phase 1 testing (up to T13-03) done.

9.2.6.4 Procedure

(a) Run tally13. Run vtact. (to activate tally13.) Run dir c: (so tally13 counts some commands).
(b) Run test-hdl THDL-03 USR HOST r HDD1 HDD2 … Check log file.

9.2.6.5 Expected results

(a) n/a
(b) Test-hdl logs that count(s) are not zero, then exits. [STV-AM-14 (exit if initial count nonzero)]

9.2.7 Test Case THDL-04

9.2.7.1 Purpose

Check that test-hdl logs command line [STV-AM-10] and disk drive [STV-AM-11] information. Check that test-hdl commands tally13 to active mode [STV-AM-13]. Check that test-hdl sends specified commands [STV-AM-15], logs each command sent [STV-AM-16], logs commands blocked [STV-AM-17], and summarizes blocking [STV-AM-18].

9.2.7.2 Setup

(a) Standard computer setup.
(b) Run tally13.

9.2.7.3 Dependency

Phase 1 testing (up to T13-03) done.

9.2.7.4 Procedure

(a) Run test-hdl THDL-04 HOST USER w HDD1 HDD2 … [STV-AM-10 (log command line), STV-AM-11 (log disk drive information), STV-AM-16 (log each

command sent), STV-AM-17 (log commands blocked), and STV-AM-18 (summarize blocking)]
(b) Run `dir a:` (to flush cache). Run `dir c:`. [STV-AM-13 (activate tally13)]
(c) Run vtreport. [STV-AM-15 (send specified commands)]

9.2.7.5 Expected results

(a) Test-hdl reports command line information and logs disk information. Test-hdl logs that write commands were sent.
(b) Results are consistent with commands to disk being blocked.
(c) Vtreport shows that test-hdl sent the write commands.

9.2.8 Test Case THDL-05

9.2.8.1 Purpose

Check that test-hdl sends specified commands [STV-AM-15], logs each command sent [STV-AM-16], logs commands blocked [STV-AM-17], summarizes blocking [STV-AM-18], and logs commands received, but not sent [STV-AM-19].

9.2.8.2 Setup

(a) Standard computer setup.
(b) Run tally13.
(c) Run vtblksom.

9.2.8.3 Dependency

Phase 1 testing (up to T13-03) done.

9.2.8.4 Procedure

(a) Run test-hdl THDL-05 HOST USER a HDD1 HDD2 … [STV-AM-16 (log each command sent), STV-AM-17 (log commands blocked), STV-AM-18 (summarize blocking), and STV-AM-19 (log unsent commands received)]
(b) Run vtreport. [STV-AM-15 (send specified commands)]

9.2.8.5 Expected results

(a) Test-hdl logs that all commands were sent. It also reports that some commands were blocked and some commands were received, but not sent.
(b) Vtreport shows proper counts.

9.2.9 Test Case THDL-06

9.2.9.1 Purpose

Check that test-hdl logs commands blocked [STV-AM-17] and summarizes blocking [STV-AM-18].

9.2.9.2 Setup

(a) Standard computer setup.

(b) Run tally13.
(c) Run vtblock.

9.2.9.3 Dependency

Phase 1 testing (up to T13-03) done.

9.2.9.4 Procedure

(a) Run test-hdl THDL-06 HOST USER a HDD1 HDD2 … [STV-AM-17 (log commands blocked) and STV-AM-18 (summarize blocking)]

9.2.9.5 Expected results

(a) Test-hdl logs that all commands were sent. It also reports that all commands were blocked.

Phase 3 validates t-off and sig-log. Since tally13 will have been validated, it can be used to validate t-off.

9.2.10 Code Review TOFF-01

9.2.10.1 Purpose

Collect information about the source code to check that the right information is reported [STV-AM-20]. Review the source code to find possible errors and to check that it meets all assertions [STV-AM-20 through STV-AM-23].

9.2.10.2 Setup

(a) Get the source code for t-off.
(b) For each reviewer, get a copy of the source code, a copy of the error checklist, and a copy of the assertions.

9.2.10.3 Dependencies

None.

9.2.10.4 Procedure

(a) Find file names, version numbers, and date and time of creation of source files.
(b) Review the code. In addition to the error checklist, check that it satisfies STV-AM-20 through STV-AM-22.

9.2.10.5 Expected results

(a) The file names, version numbers, and date and time of creation of source files.
(b) Possible violations of the error checklist and assertions STV-AM-20 through STV-AM-22.

9.2.11 Test Case TOFF-02

9.2.11.1 Purpose

Check that t-off reports file information and date and time run [STV-AM-20 and STV-AM-21]. Check that t-off issues a message and exits if tally13 is not running [STV-AM-22].

9.2.11.2 Setup

(a) Standard computer setup. Note: tally13 not running.

9.2.11.3 Dependencies

None.

9.2.11.4 Procedure

(a) Note system date and time.
(b) Run t-off TOFF-02 USR HOST. [STV-AM-20 (log compile and run information), STV-AM-21 (log command line), and STV-AM-22 (issue message and exit)]

9.2.11.5 Expected results

(a) n/a
(b) T-off logs program name, file information, and system time. It also logs command line information. T-off issues a message and exits.

9.2.12 Test Case TOFF-03

9.2.12.1 Purpose

Check that t-off commands tally13 to passive mode [STV-AM-23].

9.2.12.2 Setup

(a) Standard computer setup.
(b) Run tally13.
(c) Run vtact. (to activate tally13)
(d) Run `dir a:` (to flush cache). Run `dir c:`. (check that commands are blocked.)

9.2.12.3 Dependencies

Phase 1 testing (up to T13-03) done.

9.2.12.4 Procedure

(a) Run t-off TOFF-03 USR HOST.
(b) Run `dir a:` (to flush cache). Run `dir c:`. [STV-AM-23 (make tally13 passive)]

9.2.12.5 Expected results

(a) n/a
(b) Results are consistent with the disk being readable.

9.2.13 Code Review SIGL-01

9.2.13.1 Purpose

Collect information about the source code to check that the right information is reported [STV-AM-24]. Review the source code to find possible errors and to check that it meets all assertions [STV-AM-24 through STV-AM-27].

9.2.13.2 Setup

(a) Get the source code for sig-log.
(b) For each reviewer, get a copy of the source code, a copy of the error checklist, and a copy of the assertions.

9.2.13.3 Dependencies

None.

9.2.13.4 Procedure

(a) Find file names, version numbers, and date and time of creation of source files.
(b) Review the code. In addition to the error checklist, check that it satisfies STV-AM-24 through STV-AM-27.

9.2.13.5 Expected results

(a) The file names, version numbers, and date and time of creation of source files.
(b) Possible violations of the error checklist and assertions STV-AM-24 through STV-AM-27.

9.2.14 Test Case SIGL-02

9.2.14.1 Purpose

Check that sig-log logs file information [STV-AM-24] and date and time run [STV-AM-25]. Check that sig-log logs when the operator reports a signal [STV-AM-26].

9.2.14.2 Setup

(a) Standard computer setup.

9.2.14.3 Dependencies

None.

9.2.14.4 Procedure

(a) Note system date and time.
(b) Run sig-log SIGL-02 HOST USER. Answer that a signal was observed.

9.2.14.5 Expected results

(a) n/a
(b) Sig-log logs program name, file information, and system time, and that a signal was observed [STV-AM-24 (log file info), STV-AM-25 (log run time), STV-AM-26 (log positive response)].

9.2.15 Test Case SIGL-03

9.2.15.1 Purpose

Check that sig-log logs when the operator reports no signal [STV-AM-27].

9.2.15.2 Setup

(a) Standard computer setup.

9.2.15.3 Dependencies

None.

9.2.15.4 Procedure

(a) Run sig-log SIGL-03 HOST USER. Answer that no signal was observed.

9.2.15.5 Expected results

(a) Sig-log logs that no signal was observed [STV-AM-27 (log negative response)].

Appendix A Abbreviations and Acronyms

CFTT NIST Computer Forensic Tool Testing program
NIST U.S. National Institute of Standards and Technology
SWB Software Write Block

Appendix B Error Checklist for Code Review

B.1 Acknowledgements

Ideas for checklist questions came from these sources.

John T. Baldwin, *An Abbreviated C++ Code Inspection Checklist*, Oct 1992, www.literateprogramming.com/Baldwin-inspect.pdf, accessed 19 May 2004.
Jorge Rady de Almeida Jr., João Batista Camargo Jr., Bruno Abrantes Basseto, and Sérgio Miranda Paz, *Best Practices in Code Inspection for Safety-Critical Software*, IEEE Software, 20(3):56-63, May/June 2003.
Java Code Inspection Checklist, www.isys.uni-klu.ac.at/ISYS/Courses/03WS/sete/literatur/L06-1, accessed 20 May 2004.
John Noll, *Code Inspection Checklist*, Jan 2004, www.cse.scu.edu/~jnoll/286/projects/checklist.html, accessed 19 May 2004.

B.2 Error Checklist

Review Date _____ Reviewer _____

Name of Code (function or file) _____

1 Data & Variables

1.1 Possible uninitialized variable? No ☐

1.2 Possible off-by-1 error in array indexing? No ☐

1.3 Possible array access out of bounds (or buffer overflow)? No ☐

1.4 Can a string *not* be null-terminated? No ☐

2 Calls & Returns

2.1 Wrong parameter order or type across call or return? No ☐

2.2 Parameter doesn't match format in *printf() or *scanf()? No ☐

2.3 Returned structures on stack? No ☐

2.4 Error return from function not checked? No ☐

3 Control Flow

3.1 Switch case without break (or return)? No ☐

3.2 Switch without default? No ☐

3.3 Possible infinite loop? No ☐

3.4 Incorrect comparison or Boolean operators (eg & vs. &&)? No ☐

4 Files

4.1 Possible reuse of temporary or working files? No ☐

Describe the location and nature of possible errors.

Appendix C Source Code for Validation Programs

C.1 Vtpass

```
static char *SCCS_ID[] = {"@(#) VTPASS.CPP Version 1.3 Created 07/20/04 at 10:21:28",
                      __DATE__, __TIME__};

/*****************************************************************************
The software provided here is released by the National
Institute of Standards and Technology (NIST), an agency of
the U.S. Department of Commerce, Gaithersburg MD 20899,
USA.  The software bears no warranty, either expressed or
implied. NIST does not assume legal liability nor
responsibility for a User's use of the software or the
results of such use.

Please note that within the United States, copyright
protection, under Section 105 of the United States Code,
Title 17, is not available for any work of the United
States Government and/or for any works created by United
States Government employees. User acknowledges that this
software contains work which was created by NIST employees
and is therefore in the public domain and not subject to
copyright.  The User may use, distribute, or incorporate
this software provided the User acknowledges this via an
explicit acknowledgment of NIST-related contributions to
the User's work. User also agrees to acknowledge, via an
explicit acknowledgment, that any modifications or
alterations have been made to this software before
redistribution.
*****************************************************************************/

/***** Author: Dr. Paul E. Black, NIST/SDCT/SQG ****/
// derived from t-off.cpp Version 1.1 Created 08/02/03 at 16:24:48
// *created  "Mon May 24          2004" *by "Paul E. Black"
// *modified "Tue Jul 20 08:59:52 2004" *by "Paul E. Black"

/*****************************************************************************
*
*  vtpass is used to validate the Software Write Block test tools, in
*  particular the TALLY13 TSR program.
*
*  vtpass turns off the TALLY13 monitor and blocking function.
*
*****************************************************************************/

# include <stdio.h>
# include <stdlib.h>

/*****************************************************************************
The main routine
*****************************************************************************/

int main (int np, char **p)
{
        //*********************************************************
        // display identifying information
        (void)printf("CMD: %s", p[0]);
        for (int j = 1; j < np; j++) {
                (void)printf(" %s", p[j]);
        }
        (void)printf("\n");
        // print version and compile date and time
        (void)printf("Version: %s\n", SCCS_ID[0]);
        (void)printf("Compiled on %s at %s\n", SCCS_ID[1], SCCS_ID[2]);

        // Deactivate tally13
```

```
        asm {
                mov     dl,0
                int     0x17
        }
        return 0;
}

// end of VTPASS.CPP
```

C.2 Vtact

```
static char *SCCS_ID[] = {"@(#) VTACT.CPP Version 1.3 Created 07/20/04 at 10:21:28",
                        __DATE__, __TIME__};

/*******************************************************************************
The software provided here is released by the National
Institute of Standards and Technology (NIST), an agency of
the U.S. Department of Commerce, Gaithersburg MD 20899,
USA.  The software bears no warranty, either expressed or
implied. NIST does not assume legal liability nor
responsibility for a User's use of the software or the
results of such use.

Please note that within the United States, copyright
protection, under Section 105 of the United States Code,
Title 17, is not available for any work of the United
States Government and/or for any works created by United
States Government employees. User acknowledges that this
software contains work which was created by NIST employees
and is therefore in the public domain and not subject to
copyright.  The User may use, distribute, or incorporate
this software provided the User acknowledges this via an
explicit acknowledgment of NIST-related contributions to
the User's work. User also agrees to acknowledge, via an
explicit acknowledgment, that any modifications or
alterations have been made to this software before
redistribution.
*******************************************************************************/

/***** Author: Dr. Paul E. Black, NIST/SDCT/SQG ****/
// derived from vtpass.cpp
// *created  "Mon May 24 16:16    2004" *by "Paul E. Black"
// *modified "Tue Jul 20 09:02:12 2004" *by "Paul E. Black"

/*******************************************************************************
*
*  vtact is used to validate the Software Write Block test tools, in
*  particular the TALLY13 TSR program.
*
*  vtact turns on the TALLY13 monitor and blocking function.
*
*******************************************************************************/

# include <stdio.h>
# include <stdlib.h>
# include "wb-defs.h"

/*******************************************************************************
The main routine
*******************************************************************************/

int main (int np, char **p)
{
        unsigned int           is_active; /* indicates that tally13 TSR is running */

        //*********************************************************
        // display identifying information
        (void)printf("CMD: %s", p[0]);
        for (int j = 1; j < np; j++) {
```

```
                (void)printf(" %s", p[j]);
        }
        (void)printf("\n");
        // print version and compile date and time
        (void)printf("Version: %s\n", SCCS_ID[0]);
        (void)printf("         %s\n", WB_DEFS_SCCS_ID);
        (void)printf("Compiled on %s at %s\n", SCCS_ID[1], SCCS_ID[2]);

        // Activate tally13
        asm {
                mov     dl,1
                int     0x17
                mov     is_active,ax
        }
        if (is_active != tally13_TSR_active){
                printf ("Tally13 identifying code not returned\n");
                return 1;
        }
        printf ("Identifying code for tally13 was returned\n");
        return 0;
}

// end of VTACT.CPP
```

C.3 Vtcmdgrp

```
static char *SCCS_ID[] = {"@(#) VTCMDGRP.CPP Version 1.3 Created 07/20/04 at 10:21:28",
                        __DATE__, __TIME__};

/*****************************************************************************
The software provided here is released by the National
Institute of Standards and Technology (NIST), an agency of
the U.S. Department of Commerce, Gaithersburg MD 20899,
USA.  The software bears no warranty, either expressed or
implied. NIST does not assume legal liability nor
responsibility for a User's use of the software or the
results of such use.

Please note that within the United States, copyright
protection, under Section 105 of the United States Code,
Title 17, is not available for any work of the United
States Government and/or for any works created by United
States Government employees. User acknowledges that this
software contains work which was created by NIST employees
and is therefore in the public domain and not subject to
copyright.  The User may use, distribute, or incorporate
this software provided the User acknowledges this via an
explicit acknowledgment of NIST-related contributions to
the User's work. User also agrees to acknowledge, via an
explicit acknowledgment, that any modifications or
alterations have been made to this software before
redistribution.
*****************************************************************************/

/***** Author: Dr. Paul E. Black, NIST/SDCT/SQG ****/
// derived from test-hdl
// *created  "Tue May 25 16:32    2004" *by "Paul E. Black"
// *modified "Tue Jul 20 09:01:52 2004" *by "Paul E. Black"

/*****************************************************************************
*
*  vtcmdgrp is used to validate the Software Write Block test tools, in
*  particular the TALLY13 TSR program.
*
*  vtcmdgrp sends interrupt 0x13 commands
*
*  The logical design of vtcmdgrp is as follows:
*       check command line parameters
*               the bit position to turn on
```

```
*              the disk drive
*       If a problem is found, print a message and exit
*       Send every command
*
*****************************************************************************/

# include <stdio.h>
# include <stdlib.h>

/*****************************************************************************

The main routine
*****************************************************************************/

static const char *usageMsg = "Usage: vtcmdgrp cmdBit drive";

// the first hard disk drive is 0x80, second is 0x81, etc.
static const int firstDrive = 0x80;

int main (int argc, char **argv)
{
        int             bitN,                   /* bit to test */
                                driveN;     /* disk drive to send commands to (0-n) */

        if (argc < 2) {
                (void)printf("%s\n", usageMsg);
                return 0;
        }

        /* get N, bit to test */
        if (sscanf(argv[1], "%d", &bitN) < 1 || bitN < 0 || bitN > 7) {
                (void)printf("cmdBit (bit to test) must be between 0 and 7 inclusive\n");
                (void)printf("%s\n", usageMsg);
                return 0;
        }

        /* get disk drive number */
        if (sscanf(argv[2], "%d", &driveN) < 1 || driveN < 0 || driveN > 4) {
                (void)printf("drive must be between 0 and 4 inclusive\n");
                (void)printf("%s\n", usageMsg);
                return 0;
        }

        //*********************************************************
        // display identifying information
        (void)printf("CMD: %s", argv[0]);
        for (int j = 1; j < argc; j++) {
                (void)printf(" %s", argv[j]);
        }
        (void)printf("\n");
        // print version and compile date and time
        (void)printf("Version: %s\n", SCCS_ID[0]);
        (void)printf("Compiled on %s at %s\n", SCCS_ID[1], SCCS_ID[2]);

        // send all 128 commands with bitN on.  For programming ease, generate
        // all 256 possible codes, then set bitN on.
        int bitNmask = 1 << bitN;
        for (int command = 0; command < 256; command++) {
                int cmdNset = command | bitNmask,     // command with bit N on
                        fun = cmdNset << 8,                                 // put command
code in upper byte
                        drive = driveN + firstDrive;        // hard drive starting at
0x80

                // send a command to the int 13 interface
                asm {
                                push    si
                                mov     ax,fun
                                mov     dx,drive
                                int     0x13
                                pop     si
```

```
                }
        }

        return 0;
}

// end of VTCMDGRP.CPP
```

C.4 Vtreport

```
static char *SCCS_ID[] = {"@(#) VTREPORT.CPP Version 1.3 Created 07/20/04 at 10:21:28",
                      __DATE__, __TIME__};

/*****************************************************************************
The software provided here is released by the National
Institute of Standards and Technology (NIST), an agency of
the U.S. Department of Commerce, Gaithersburg MD 20899,
USA.  The software bears no warranty, either expressed or
implied. NIST does not assume legal liability nor
responsibility for a User's use of the software or the
results of such use.

Please note that within the United States, copyright
protection, under Section 105 of the United States Code,
Title 17, is not available for any work of the United
States Government and/or for any works created by United
States Government employees. User acknowledges that this
software contains work which was created by NIST employees
and is therefore in the public domain and not subject to
copyright.  The User may use, distribute, or incorporate
this software provided the User acknowledges this via an
explicit acknowledgment of NIST-related contributions to
the User's work. User also agrees to acknowledge, via an
explicit acknowledgment, that any modifications or
alterations have been made to this software before
redistribution.
*****************************************************************************/

/***** Author: Dr. Paul E. Black, NIST/SDCT/SQG ****/
// derived from test-hdl
// *created  "Thu Jun  3 09:26    2004" *by "Paul E. Black"
// *modified "Tue Jul 20 09:01:28 2004" *by "Paul E. Black"

/*****************************************************************************
*
*  vtreport is used to validate the Software Write Block test tools, in
*  particular the TALLY13 TSR program.
*
*  vtreport queries tally13 for interrupt 0x13 commands received
*
*  The logical design of vtcmdgrp is as follows:
*       check command line parameters
*               for each possible drive
*                       for each possible command
*                               query tally13 for number of commands received
*                               report count
*
*****************************************************************************/

# include <stdio.h>
# include <stdlib.h>

/*****************************************************************************

The main routine
*****************************************************************************/

static const char *usageMsg = "Usage: vtreport";
```

```
// the first hard disk drive is 0x80, second is 0x81, etc.
static const int firstDrive = 0x80;

// check maximum number of drives tally13 can handle
static const int numberOfDrives = 5;

int main (int argc, char **argv)
{
	if (argc != 1) {
		(void)printf("%s\n", usageMsg);
		return 0;
	}

	//*******************************************************
	// display identifying information
	(void)printf("CMD: %s", argv[0]);
	for (int j = 1; j < argc; j++) {
		(void)printf(" %s", argv[j]);
	}
	(void)printf("\n");
	// print version and compile date and time
	(void)printf("Version: %s\n", SCCS_ID[0]);
	(void)printf("Compiled on %s at %s\n", SCCS_ID[1], SCCS_ID[2]);

	// for each drive, for each command, get current count
	for (int driveNo = 0; driveNo < numberOfDrives; driveNo++) {
		int drive = driveNo + firstDrive;
		(void)printf("Drive %d (%x)\n", driveNo, drive);
		 // column headings for count report
		(void)printf("Command 0 1 2 3 4 5 6 7 8 9 a b c d e f 0 1 2 3 4 5 6 7 8 9
a b c d e f\n");
		for (int j = 0; j < 256; j++){ // for each command code
			int fun = j << 8; // shift to upper byte
			int ans = 1776; // arbitrary, unusual initial value to check
assignment

			asm {
				push si
				mov ax,fun
				mov dx,drive
				int 0x17
				mov ans,cx
				pop si
			}

			static const int countsPerLine = 32;
			// print command code at start of every line
			if ((j % countsPerLine) == 0) {
				(void)printf("     %02x", j);
			}
			(void)printf("%2d", ans);
			// after last count, start a new line
			if ((j % countsPerLine) == countsPerLine-1) {
				(void)printf("\n");
			}
		}
	}

	return 0;
}

// end of VTREPORT.CPP
```

C.5 Vtblksom

```
;************************************************
; The software provided here is released by the National
; Institute of Standards and Technology (NIST), an agency of
; the U.S. Department of Commerce, Gaithersburg MD 20899,
```

```
; USA.  The software bears no warranty, either expressed or
; implied. NIST does not assume legal liability nor
; responsibility for a User's use of the software or the
; results of such use.
;
; Please note that within the United States, copyright
; protection, under Section 105 of the United States Code,
; Title 17, is not available for any work of the United
; States Government and/or for any works created by United
; States Government employees. User acknowledges that this
; software contains work which was created by NIST employees
; and is therefore in the public domain and not subject to
; copyright.  The User may use, distribute, or incorporate
; this software provided the User acknowledges this via an
; explicit acknowledgment of NIST-related contributions to
; the User's work. User also agrees to acknowledge, via an
; explicit acknowledgment, that any modifications or
; alterations have been made to this software before
; redistribution.
;*************************************************
;       vtblksom -- block all hard disk BIOS Int 0x13 requests
;               greater than 0x40 (arbitrary cutoff).  Change all
;               0x80 (arbitrary) commands to 0x81 and pass on.
;       based on vtblock
;       Author: Dr. Paul E. Black
; *created  "Wed Jun 17 10:45:29 2004" *by "Paul E. Black"
; *modified "Tue Jul 20 08:52:10 2004" *by "Paul E. Black"
;       Usage: vtblksom
;*************************************************
                        .model  tiny
                        version m510
                        P386
.code
cr                      equ             0ah
lf                      equ             0dh
doscall         equ             21h
dos_tsr         equ             3100h
set_i13         equ             2513h
get_i13         equ             3513h
print_cmd               equ             9h
;*************************************************
; Print a message
;*************************************************
print           MACRO   message
                        lea             dx,message
                        mov             ah,print_cmd
                        int             21h
                        ENDM


;*************************************************
;*************************************************
decode  MACRO   from ; from is a byte register with the digit pair
        mov     AH,from ; move digits to AH
        shr     AX,4    ; shift left digit into low bits & right digit into AL
        shr     AL,4    ; move right digit into low bits
        or      AX,3030H ; convert to ASCII, leave in AX
        ENDM
get_date MACRO string  ; get a date, put in string
        mov             AH,04H  ; setup for date BIOS service
        int             1AH     ; get the date mm in DH
        decode          DH      ; decode the month
        mov             string,AH       ; save month in string
        mov             string+1,AL
        decode          DL              ; day is in DL
        mov             string+3,AH     ; save day of month
        mov             string+4,AL
        decode          CL              ; year is in CL
        mov             string+6,AH     ; save year digits
        mov             string+7,AL
        ENDM
```

```
get_time MACRO string                   ; get current time from BIOS
        mov             AH,02H          ; setup for time service
        int             1AH             ; get the time
        decode          CH              ; hours is in CH
        mov             string,AH       ; save hours
        mov             string+1,AL
        decode          CL              ; minutes is in CL
        mov             string+3,AH     ; save minutes
        mov             string+4,AL
        decode          DH              ; seconds is in DH
        mov             string+6,AH     ; save seconds
        mov             string+7,AL
        ENDM
;*************************************************
;*************************************************
start:
                        jmp             install
;*************************************************
;       DISK REQUEST
;               AL      MBZ for write command
;               AH      Command, 42 = read, 43 = write
;               DL      Drive ID
;               DS      DAP segment
;               SI      DAP offset
;*************************************************
;       DISK ADDRESS PACKET (DAP)
;               0       Must be 16
;               1       MBZ
;               2       # of sectors to read/write
;               3       MBZ
;               4-5     Data area segment
;               6-7     Data area offset
;               8-      Sector LBA (64 bit value)
;*************************************************

change_func:
                        inc             ah              ; change function
use_prev_service:
                        popf                            ; restore flags
                        jmp             CS:bios_old     ; execute previous BIOS service
tally_service:
                        pushf                           ; save flags
                        cmp             dl,7Fh
                        jbe             use_prev_service                ; not hard drive
                        cmp             ah,40h
                        jbe             use_prev_service                ; func <= 0x40; pass
it
                        cmp             ah,80h
                        je              change_func                     ; func is 0x80; change
it
                        popf                            ; restore flags

                        iret


;*************************************************
; Resident data area
;*************************************************
bios_old        label   dword
biosoff         dw              0
biosseg         dw              0


install:
                        mov             ax,@data                        ; put address of data
segment ...
                        mov             ds,ax                           ; ... in DS

                        get_date date                           ; add date to signoff line
                        get_time time                           ; add time to signoff line
```

```
                        mov             ax,get_i13                      ; get current DISK
BIOS interrupt vector
                        int             doscall

                        mov             biosoff,bx                      ; save offset of DISK
handler
                        mov             biosseg,es                      ; and segment

                        lea             dx,tally_service                ; address of new disk
service
                        mov             ax,set_i13                      ; install as new BIOS
interrupt vector
                        int             doscall

                        print   signon                          ; print sign on and sign off
message

                        lea             dx,install                      ; calculate resident
part size
                        add             dx,110h                 ; don't forget PSP
                        mov             cx,4
                        shr             dx,cl                           ; size is in
paragraphs!!!
                        inc             dx                              ; round up by 1
                        mov             ax,dos_tsr                      ; get ready to TSR
;;;;mov          ah,4ch                  ; DOS terminate program
;;;;mov          al,0                            ; return code 0
                        int             doscall                 ; Now TSR


;*************************************************
; non-resident data area
;*************************************************

signon          db              'Block or change some BIOS interrupt 13h (disk
service)',cr,lf
                        db              ??filename,' compiled on '
                        db              ??date,' at ',??time,cr,lf
                        db              '@(#) vtblksom.asm Version 1.2 Created 07/20/04 at
10:21:28',cr,lf
signoff         db              'Now ('
date            db              '99/99/99 at '
time            db              '99:99:99) Going . . .  TSR',cr,lf,'$'
                        end             start
```

C.6 Vtblock

```
;*************************************************
; The software provided here is released by the National
; Institute of Standards and Technology (NIST), an agency of
; the U.S. Department of Commerce, Gaithersburg MD 20899,
; USA.  The software bears no warranty, either expressed or
; implied. NIST does not assume legal liability nor
; responsibility for a User's use of the software or the
; results of such use.
;
; Please note that within the United States, copyright
; protection, under Section 105 of the United States Code,
; Title 17, is not available for any work of the United
; States Government and/or for any works created by United
; States Government employees. User acknowledges that this
; software contains work which was created by NIST employees
; and is therefore in the public domain and not subject to
; copyright.  The User may use, distribute, or incorporate
; this software provided the User acknowledges this via an
; explicit acknowledgment of NIST-related contributions to
; the User's work. User also agrees to acknowledge, via an
; explicit acknowledgment, that any modifications or
```

```
; alterations have been made to this software before
; redistribution.
;*************************************************
;       vtblock -- block all hard disk BIOS Int 0x13 requests
;       based on tally13
;       Author: Dr. Paul E. Black
; *created  "Wed Jun 16 14:27:29 2004" *by "Paul E. Black"
; *modified "Tue Jul 20 08:51:38 2004" *by "Paul E. Black"
;       Usage: vtblock
;*************************************************
                        .model  tiny
                        version m510
                        P386
.code
cr                      equ             0ah
lf                      equ             0dh
doscall         equ             21h
dos_tsr         equ             3100h
set_i13         equ             2513h
get_i13         equ             3513h
print_cmd               equ             9h
;*************************************************
; Print a message
;*************************************************
print           MACRO   message
                        lea             dx,message
                        mov             ah,print_cmd
                        int             21h
                        ENDM


;*************************************************
;*************************************************
decode  MACRO   from ; from is a byte register with the digit pair
        mov     AH,from ; move digits to AH
        shr     AX,4    ; shift left digit into low bits & right digit into AL
        shr     AL,4    ; move right digit into low bits
        or      AX,3030H ; convert to ASCII, leave in AX
        ENDM
get_date MACRO string  ; get a date, put in string
        mov             AH,04H  ; setup for date BIOS service
        int             1AH     ; get the date mm in DH
        decode          DH      ; decode the month
        mov             string,AH       ; save month in string
        mov             string+1,AL
        decode          DL              ; day is in DL
        mov             string+3,AH     ; save day of month
        mov             string+4,AL
        decode          CL              ; year is in CL
        mov             string+6,AH     ; save year digits
        mov             string+7,AL
        ENDM

get_time MACRO string                   ; get current time from BIOS
        mov             AH,02H          ; setup for time service
        int             1AH             ; get the time
        decode          CH              ; hours is in CH
        mov             string,AH       ; save hours
        mov             string+1,AL
        decode          CL              ; minutes is in CL
        mov             string+3,AH     ; save minutes
        mov             string+4,AL
        decode          DH              ; seconds is in DH
        mov             string+6,AH     ; save seconds
        mov             string+7,AL
        ENDM
;*************************************************
;*************************************************
start:
                        jmp             install
```

```
not_hard_drive:
                        popf                            ; restore flags
                        jmp             CS:bios_old     ; execute previous BIOS service
tally_service:
                        pushf                           ; save flags
                        cmp             dl,7Fh
                        jbe             not_hard_drive          ; no match, not hard drive
                        popf                            ; restore flags

                        iret


;*************************************************
; Resident data area
;*************************************************
bios_old        label   dword
biosoff         dw              0
biosseg         dw              0


install:
                        mov             ax,@data                ; put address of data segment
...
                        mov             ds,ax                   ; ... in DS

                        get_date date           ; add date to signoff line
                        get_time time           ; add time to signoff line

                        mov             ax,get_i13                      ; get current DISK
BIOS interrupt vector
                        int             doscall

                        mov             biosoff,bx                      ; save offset of DISK
handler
                        mov             biosseg,es                      ; and segment

                        lea             dx,tally_service                ; address of new disk
service
                        mov             ax,set_i13                      ; install as new BIOS
interrupt vector
                        int             doscall

                        print   signon                          ; print sign on and sign off
message

                        lea             dx,install                      ; calculate resident
part size
                        add             dx,110h                 ; don't forget PSP
                        mov             cx,4
                        shr             dx,cl                           ; size is in
paragraphs!!!
                        inc             dx                              ; round up by 1
                        mov             ax,dos_tsr                      ; get ready to TSR
;;;;mov ah,4ch ; DOS terminate program
;;;;mov al,0   ; return code 0
                        int             doscall                 ; Now TSR


;*************************************************
; non-resident data area
;*************************************************

signon          db              'Block BIOS interrupt 13h (hard disk service)',cr,lf
                        db              ??filename,'compiled on '
                        db              ??date,' at ',??time,cr,lf
                        db              '@(#) vtblock.asm Version 1.2 Created 07/20/04 at
10:21:28',cr,lf
signoff         db              'Now ('
date            db              '99/99/99 at '
time            db              '99:99:99) Going . . .  TSR',cr,lf,'$'
                        end             start
```