Universidade Federal do Ceará
Departamento de Engenharia de Teleinformática
Curso de Graduação em Engenharia de Teleinformática

Patrícia Jamile de Oliveira Martins

Framework de Diagnóstico de Discos Rígidos em Sistemas Linux

Fortaleza – Ceará
Dezembro 2011

Autor:
Patrícia Jamile de Oliveira Martins

Orientador:
Prof. Msc. Ricardo Jardel Nunes da Silveira

Framework de Diagnóstico de Discos Rígidos em Sistemas Linux

Monografia de Conclusão de Curso apresentada à Coordenação do Curso de Pós-Graduação em Engenharia de Teleinformática da Universidade Federal do Ceará como parte dos requisitos para obtenção do grau de **Engenheira de Teleinformática**.

Fortaleza – Ceará
Fevereiro 2011

Patrícia Jamile de Oliveira Martins

Framework de Diagnóstico de Discos Rígidos em Sistemas Linux

Esta Monografia foi julgada adequada para a obtenção do diploma de Engenheira do Curso de Graduação em Engenharia de Teleinformática da Universidade Federal do Ceará.

Patrícia Jamile de Oliveira Martins

Banca Examinadora:

Prof. Msc. Ricardo Jardel Nunes da Silveira
Orientador

Prof. Helano de Sousa Castro

Prof. Alexandre Augusto da Penha Coelho

Prof. Jarbas Aryel Nunes da Silveira

Fortaleza, 6 de fevereiro de 2013

Resumo

Discos rígidos são elementos fundamentais em qualquer sistema computacional moderno e é importante monitorar a "saúde" destes dispositivos, pois são utilizados desde computadores pessoais a aplicações de risco. Este trabalho tem como objetivo construir um *framework* em sistema Linux, utilizando comandos dos padrões *Advanced Technology Attachment* (ATA) e *Small Computer System Interface* (SCSI), para dar suporte ao desenvolvimento de algoritmos de teste de discos rígidos. Para demonstrar o uso do framework, são implementados alguns algoritmos de teste rápido, testes que variam entre 5 e 10 minutos, que apresentaram resultados comparáveis com ferramentas de diagnóstico do mercado. Por último estes algoritmos são embarcados em uma versão inicializável do Linux, o que permite realizar testes em quais quer computadores que suportem inicialização por *pendrive* ou *live cd*, independentemente do sistema operacional instalado. Os resultados dos testes dos algoritmos foram bastante satisfatórios, demonstrando a eficiência do *framework*.

Para o desenvolvimento do trabalho todos os comandos foram desenvolvidos em linguagem C++ ANSI. Vários comandos ATA e SCSI foram implementados para dar suporte a leitura, auto-teste e outras funcionalidades e a partir deles os algoritmos são implementados.

Palavras-chaves: Framework de Diagnóstico, Discos Rígidos, HDD, SSD, SCSI, ATA.

Abstract

Hard disks are very important elements in any modern computer system and it is advisable to monitor the health of these devices because they are used for applications from personal computers to risk activities. This work aims to build a framework in Linux system using the standard commands ATA and SCSI to support the development of algorithms for test drives. To demonstrate the use of the framework, algorithms are implemented some test that is compared with diagnostic tools on the market. Finally, these algorithms are embedded in a bootable version of Linux, which allows you to perform during any computers that support booting from a live CD or USB stick, regardless of operating system installed. The test results show the algorithm is very satisfactory, demonstrating the effectiveness of the framework.

For the development of the work all commands have been developed in ANSI C + + language. Several commands ATA and SCSI were implemented to support reading, self-testing and other features. Also, the algorithm are implemented from them. **Keywords: Diagnostic Framework, Hard Disk, Solid-State Drive, Hard Disk Drive, SCSI, ATA, Linux** .

Dedico este trabalho primeiramente a Deus,
à minha mãe Graça, ao meu pai Genésio,
à minha família, ao meu namorado Jefferson,
aos professores, aos amigos e
aos colegas de trabalho que me acompanharam
durante a minha formação

... Lembre-se do valor da imaginação, e da noção de que quando você olha ao redor, tudo que você vê já foi imaginado em algum momento por alguém.

Neil Gaiman

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

ABS *Air Bearing Heads*

ANSI *American National Standards Institute*

ATA *Advanced Technology Attachment*

ATAPI *Advanced Technology Attachment Packet Interface*

BD *Blu-Ray Disk*

BIOS *Basic Input/Output System*

CD *Compact Disk*

CD-ROM *CD - Read Only Memory*

CCROS *Charged Capacitor Read Only Store*

CHS *Cylinder-Head-Sector*

CRC *Cyclic Redundancy Check*

DVD *Digital Versatile Disk*

DASD *Direct Access Storage Devices*

DRAM *Dynamic RAM*

EIDE *Extended IDE*

EPROM *Erasable Programmable Read-Only Memory*

EEPROM *Electrically Erasable Programmable Read-Only Memory*

IBM *International Business Machines*

ISA *Industry Standard Architecture*

iSCSI *Internet SCSI*

LBA *Logical Block Addressing*

LESC Laboratório de Engenharia de Sistemas de Computação

HD *Hard Disk*

HDD *Hard Disk Drive*

HD-DVD *High Definition DVD*

IDE *Integrated Drive Electronics*

MBR *Master Boot Record*

MLC *Multi-Level Cell*

PATA *Parallel ATA*

PCB *Printed Circuit Board*

RAM *Random Access Memory*

RAMAC *Random Access Method of Accounting and Control*

ROM *Read Only Memory*

SAS *Serial Attached SCSI*

SATA *Serial ATA*

SCSI *Small Computer System Interface*

SLC *Single-Level Cell*

SMART *Self-Monitoring, Analysis and Reporting Technology*

SPI *SCSI Parallel Interface*

SSD *Solid-State Drive*

UFC Universidade Federal do Ceará

USB *Universal Serial Bus*

VCM *Voice Coil Motor*

Capítulo 1

Introdução

Vivemos em uma era da tecnologia da informação onde cada aspecto de nossa vida passa por algum tipo de processamento ou armazenamento de informações. Este armazenamento pode ser feito de maneira temporária, como nas memórias *Random Access Memory* (RAM), ou de maneira permanente, como nos discos rígidos.

Os discos rígidos desempenham um grande papel no desenvolvimento tecnológico atual. Eles são memórias não voláteis, fundamentais ao funcionamento de qualquer sistema computacional moderno, responsáveis por armazenar sistemas operacionais, programas e dados dos usuários.

Atualmente, os dois principais tipos de discos rígidos, considerados neste trabalho como os dispositivos que armazenam o sistema operacional e dados em um computador , são o HDD, baseado em armazenamento magnético, que é a tecnologia que domina o mercado, e o *Solid-State Drive* (SSD) baseado em memória *flash*, popularizada recentemente. Estas duas tecnologias seguem dois padrões definidos pela *American National Standards Institute* (ANSI), são eles: ATA[1] e SCSI. Estes padrões definem, entre outras características, os protocolos de comunicação, as conexões físicas e os comandos de controle.

Nas últimas décadas os discos rígidos passaram por grandes mudanças, tais como aumento de densidade, no qual mais dados são gravados no mesmo espaço físico, diminuição de tamanho e o acréscimo da sua capacidade de armazenamento total.

[1] ATA foi o nome dado ao padrão *Integrated Drive Electronics* (IDE) ao ser oficializado pela ANSI, ele passou a ser chamado por muitas pessoas como ATA, IDE, ou ainda IDE\ATA

Devido a este aumento da densidade, os discos rígidos estão mais sujeitos a defeitos de fabricação. Além disso, tais dispositivos sofrem desgaste natural decorrente do uso, aumentando a probabilidade de ocorrência de falhas. Estejam eles em sistemas que controlam aplicações críticas, tais como bolsa de valores, plantas de usinas nucleares ou sistemas hospitalares, ou em um *desktop* de um usuário comum, é importante que um sistema de diagnóstico eficiente seja executado periodicamente, visando evitar perda de dados, corrupção de arquivos ou falhas catastróficas, as quais podem implicar inclusive na perda de vidas humanas.

1.1 Motivação

A maioria das ferramentas de diagnóstico de dispositivos de *hardware* disponíveis no mercado foi desenvolvida para sistemas operacionais *Windows*. Em pesquisas realizadas em *sites* de *downloads*[2] por "diagnóstico de hardware" encontra-se, em média, 171 *softwares* para Windows e apenas 11 para Linux. Entretanto nestes resultados estão vários *softwares* que não tratam de discos rígidos, apenas de CPU ou medição de temperatura na placa-mãe, por exemplo. Filtrando esses resultados por "discos rígidos" encontramos apenas 75 *softwares* de diagnóstico para *Windows* e 5 para Linux.

Estes *softwares* de diagnóstico executam diferentes algoritmos de teste, para detectar erros de leitura e escrita e avaliar a "saúde" do dispositivo, se há setores com falha ou na iminência de falha, dando ao usuário tempo para realizar um *backup* ou substituir o dispositivo. Como a capacidade de armazenamento aumentou fortemente, analisar todos os setores de um disco é cada vez mais demorado, podendo levar várias horas e até dias, por esse motivo muitos destes softwares dispõem de testes rápidos que levam de 5 a 10 minutos para analisar o disco.

O *Windows* é o sistema operacional utilizado na maioria dos computadores pessoais. No entanto, Linux é o sistema operacional preferido para servidores e computadores de grande porte, responsáveis por prover os mais diversos serviços. O Linux é extensivamente utilizado desde servidores a sistemas embarcados e possui várias distribuições com diferentes propostas de uso como Fedora, SUSE, Ubuntu, Debian e *Red Hat*, por exemplo, sendo a maioria delas gratuita. Ele dispõe de código aberto, possibilitando ao desenvolvedor o amplo acesso ao sistema e a capacidade

[2]Site *www.baixaki.com.br*, 175 resultados para *Windows* e 6 para Linux. Site *www.superdownloads.com.br* 168 resultados para Windows e 17 para Linux, em Setembro de 2011

para adaptá-lo às suas necessidades. O mesmo dispõe ainda de várias funcionalidades úteis, como versões inicializáveis, que permitem a sua execução em qualquer computador cujo o *Basic Input/Output System* (BIOS) suporte inicializacação por *Pendrive* ou *Compact Disk* (CD).

Há duas maneiras de acessar um *Hard Disk* (HD) via *software*: Acessando diretamente a controladora do disco ou através de funções disponibilizadas pelo BIOS. O meio mais rápido de ter acesso ao disco é acessar diretamente a controladora, através de comandos ATA ou SCSI, mas isso requer conhecimento sobre o *hardware*, (**??**). Este normalmente é o meio utilizado pelos *softwares* de diagnóstico, que enviam comandos ATA e SCSI para realizar algoritmos de teste e também ter acesso ao SMART, que é um sistema de predição de falhas adotado pela indústria com o intuito de aumentar a confiabilidade dos discos rígidos, disponível na maioria dos HDs.

No Linux há alguns programas que implementam comandos ATA e SCSI e acessam o SMART, tais como *smartmoontools, hdparm, sg3-utils, util-linux*, entre outros mas eles dispõem de poucas opções para o usuário. Além disso, para utilizá-los em um algoritmo de teste, por exemplo, eles teriam que ser executados através de chamadas de aplicativos, o que pode levar à perda de desempenho e funcionalidade (**??**).

Neste sentido, o desenvolvimento de um *framework* para auxiliar no diagnóstico de discos rígidos é proposto neste trabalho. Ele é baseado em comandos ATA, SCSI e SMART, voltado para o desenvolvimento de testes rápidos, que levam de 5 a 10 minutos para diagnosticar um dispositivo, e para a execução em *desktops* e *laptops* em uso, e pensado para sistemas Linux. A partir deste *framework*, uma camada de aplicação com os algoritmos de testes mais importantes e mais frequentes nas ferramentas de diagnóstico do mercado é implementada, focando em algoritmos para testes rápidos. Esta camada foi portada para uma versão inicializável do Linux, chamada neste trabalho de *Bootable*.

1.2 Objetivos

Os objetivos gerais e específicos desta monografia, são apresentados a seguir.

1.2.1 Objetivos Gerais

Desenvolver um *framework* para auxiliar no desenvolvimento de testes de diagnóstico de discos rígidos em sistemas Linux para ser executado em *desktops* e *laptops*. Criar uma camada de aplicação, um programa feito a partir do *framework* desenvolvido, com a implementação de alguns algoritmos de teste. Comparar a eficiência dos algoritmos implementados na camada de aplicação com os testes de ferramentas de diagnóstico do mercado. Portar os testes criados na camada de aplicação para uma versão inicializável do Linux, um *Bootable*.

1.2.2 Objetivos Específicos

Para o desenvolvimento da monografia alguns objetivos específicos precisaram ser definidos, sendo estes listados a seguir:

i. Implementação e validação de comandos ATA e SCSI em linguagem C++.

ii. Escolha e implementação de algoritmos de teste em C++, baseados nos testes das ferramentas de diagnóstico de mercado.

iii. Criação de uma versão inicializável do Linux contendo os algoritmos desenvolvidos neste trabalho.

iv. Avaliação da eficiência dos algoritmos implementados neste trabalho em comparação com os testes realizados por ferramentas de diagnóstico.

v. Análise dos resultados.

1.3 Organização do Texto

Este texto está organizada em 6 capítulos. O capítulo 2 apresenta um apanhado sobre a história e o funcionamento das tecnologias nas quais os discos rígidos atuais se baseiam, os padrões adotados pela indústria são apresentados e um comparativo entre as tecnologias HDD e SSD é realizado. O capítulo 3 detalha os fundamentos de diagnóstico de falha, as principais falhas em discos rígidos e os padrões ATA e SCSI.

No capítulo 4, a arquitetura do *framework* desenvolvido neste trabalho é descrita, bem como os algoritmos de testes implementados, o processo de criação de uma

versão inicializável do Linux e a metodologia de testes. No capítulo 5 são descritos e discutidos os resultados obtidos no trabalho, a implementação da camada de aplicação e os resultados obtidos nos testes com os algoritmos implementados. Por fim, o último capítulo apresenta as conclusões, considerações finais e as perspectivas futuras deste trabalho.

Capítulo 2

Discos Rígidos

Neste capítulo, a evolução dos discos rígidos é descrita de maneira breve para contextualizar o leitor quanto às tecnologias utilizadas atualmente e ao seu funcionamento.

2.1 Introdução

Os sistemas modernos de computação usam diferentes tecnologias para armazenar estas informações, seja temporariamente ou permanentemente. Essas tecnologias podem ser memórias de estado sólido, como *Read Only Memory* (ROM), RAM e memórias *Flash*, armazenamento magnético, como HDD, *floppy disk* e *tape*, ou armazenamento óptico, como *CD - Read Only Memory* (CD-ROM), *Digital Versatile Disk* (DVD), *Blu-Ray Disk* (BD) e *High Definition DVD* (HD-DVD).

Os discos rígidos desempenham um grande papel neste desenvolvimento tecnológico. Atualmente, eles tanto podem ser dispositivos de armazenamento magnético como os HDDs, quanto circuitos de memória *Flash*, no caso dos SSDs atuais, ou ainda dispositivos híbridos (**??**). Os principais fatores considerados na escolha entre essas tecnologias são o custo, a taxa de transferência de dados, o tempo de acesso e a confiabilidade (**??**).

A pirâmide de hierarquia de armazenamento, Figura 2.1, descreve uma hierarquia de tecnologias de memória em função do custo, da capacidade de armazenamento e do tempo de acesso. Nela, os dois principais tipos de discos rígidos, HDD e SSD, se encontram na camada intermediária.

A região superior da pirâmide contém registradores do processador, memória

cache e memória RAM, que se caracterizam pelo alto custo, rapidez no acesso aos dados e baixa capacidade de armazenamento. Já a região intermediária contém memórias Flash, das quais são feitos os SSDs, discos magnéticos e discos ópticos, que possuem um custo menor em comparação às memórias internas, maior capacidade de armazenamento e menor velocidade no acesso aos dados. Na região inferior se encontram fitas magnéticas, dispositivos de acesso sequencial usados para cópias de segurança.

Figura 2.1: Pirâmide Hierárquica de Armazenamento, adaptada de (**??**)

2.2 Histórico

O disco rígido, também conhecido como *Direct Access Storage Devices* (DASD), em referência à capacidade de acesso direto, foi um dos dispositivos que mais evoluiu na história da computação.

Em pouco mais de 5 décadas, a indústria de armazenamento de dados evoluiu de um grande aparato de 50 discos de 24 *in* de diâmetro (aproximadamente 61 cm), com uma capacidade total de 5 MB para discos que armazenam até 4 TB em discos de 3,5 *in* de diâmetro (pouco menos de 9 cm). Esta enorme evolução se tornou possível graças a avanços em diversas áreas do conhecimento incluindo: materiais, tribologia[1], mecânica, mecatrônica, processamento de sinais e eletrônica, (**??**).

[1]Tribologia é a ciência que estuda a interação de superfícies, submetidas a cargas e a movimentos relativos.

Os primeiros HDs eram significativamente diferentes dos que usamos agora, não apenas em relação ao tamanho, mas em aspectos como capacidade de armazenamento, taxa de transferência de dados e tecnologia utilizada.

Os HDDs e SSDs são contemporâneos e foram desenvolvidos pela *International Business Machines* (IBM)[2], empresa que começou a se destacar no início do século XX através do desenvolvimento de máquinas mecânicas e elétricas, como calculadoras de repetição e máquinas datilografia, e que foi responsável por muitos avanços na computação.

As primeiras máquinas utilizavam cartões e fitas de papel para codificar e para armazenar informações, mas, entre outras desvantagens, não havia memória RAM suficiente e o *software* e dados iniciais precisavam ser carregados cada vez que a máquina era ligada.

Havia a necessidade de memórias mais eficientes, temporárias e permanentes. Então a IBM começou a desenvolver memórias alternativas, principalmente baseadas em magnetismo. Ela desenvolveu os discos magnéticos, exemplificados na Figura 2.2.a, e com a popularização dos transistores na década de 1950 ela desenvolveu o primeiro dispositivo de memória não volátil de estado sólido chamado *Charged Capacitor Read Only Store* (CCROS) (**??**), o antecessor das memórias *Erasable Programmable Read-Only Memory* (EPROM), *Electrically Erasable Programmable Read-Only Memory* (EEPROM) e *Flash*, mostrado na Figura 2.2.b.

Na mesma época, outro método de SSD foi desenvolvido usando magnetismo, era o Núcleo de Memória (*Core Memory*), que utilizava pequenos núcleos individuais de ferrite[3] unidos por cobre, como exemplificado na Figura 2.2.c.

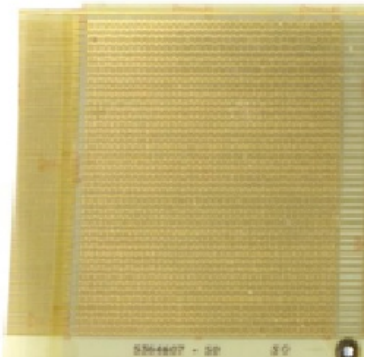
2.2.1 Primeiros Dispositivos

O primeiro equipamento a utilizar um DASD não volátil foi o *Random Access Method of Accounting and Control* (RAMAC), lançado pela IBM em 1956. A unidade de disco do RAMAC, chamada IBM 350, mostrada na Figura 2.3, continha 50 discos, cada um com 24 *in* de diâmetro e que podiam armazenar 5 MB a uma densidade de gravação de $2Kbits/in^2$ (**??**) e taxa de transferência de dados

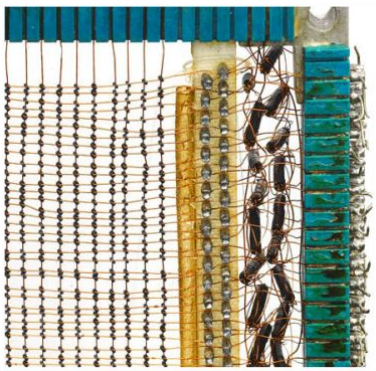
[2]A IBM foi o novo nome dado à C-T-R(Computing-Tabulating-Recording Company), empresa fruto da fusão de outras três empresas entre elas a *Tabulating Machine Company*, fundada por Herman Hollerith, especializada no processamento de dados de cartões perfurados.

[3]Ferrite, material feito de cerâmica com propriedades eletromagnéticas

(a) Disco Magnético, adaptada de (**??**)



(b) Cartão CCROS, adaptada de (**??**)



(c) Matriz de Núcleos de Memória, adaptada de (**??**)

Figura 2.2: Primeiros Dispositivos de Armazenamento

de $8.8KB/s$(**??**). Ela não possuía sistema de controle de malha fechada para posicionamento da cabeça de leitura e escrita.

Em 1961, o primeiro dispositivo HDD a usar de rolamentos de ar, *Air Bearing Heads* (ABS),(**??**), foi lançado e, em 1963, o primeiro disco removível.

Em 1964, o primeiro equipamento a utilizar cartões CCROS[4], é lançado pela IBM, era a família IBM System 360, família de *mainframes*.[5]

Em 1971, a IBM lançou o primeiro HDD com controle de malha fechada, o IBM 330 *Merlin Drive*. Ele mantinha a posição da cabeça relativa à trilha do disco monitorada por um sensor de movimento.

Em 1973, a IBM introduziu outro modelo de HDD, o IBM 3340, que usava cabeça

[4]Cartões CCROS foram os primeiro dispositivos SSD

[5]Mainframe, computador de grande porte, dedicado normalmente ao processamento de um volume grande de informações.

e braço de ferrite. Ele continha 2 ou 4 discos de 14 *in* de diâmetro, aproximadamente $35,56cm$, possuía capacidade de armazenamento de 35 MB para um dispositivos com 2 discos, taxa de transferência de dados de 0.8MB/s e densidade de gravação média de $1,69Mbits/in^2$. Este ficou conhecido como disco Winchester, o primeiro HDD a usar a malha de controle dos dispositivos atuais.

Figura 2.3: IBM 350, adaptada de (**??**)

2.2.2 Desenvolvimento dos *Desktops*

O desenvolvimento de computadores *desktop*, no início da década de 1980, representou uma nova era da tecnologia da informação. Este tipo de computador iniciou uma nova demanda na indústria de armazenamento de dados, pois os HDs, até então usados apenas para soluções corporativas, teriam de se adequar a equipamentos mais compactos.

A necessidade de DASD com tamanho físico adequado a desktops se tornou evidente e foi quando os discos de $5\frac{1}{4}$ *in* surgiram. Estes dispositivos tinham 5 MB de capacidade de armazenamento e traziam a primeira interface de disco, eram os discos da série ST506, introduzida pela empresa Seagate Technology.

A série ST506 usava motor de passo como atuador. Esse motor, ao receber um impulso elétrico, movimenta o braço por uma curta distância, correspondente ao comprimento de uma trilha. Eles eram muito suscetíveis a problemas de desalinhamento e não permitiam densidades de gravação muito altas. Logo nos primeiros anos da década de 1980, foi produzido em grande escala o primeiro

dispositivo de $5\frac{1}{4}$ *in* com atuador do tipo *Voice Coil Motor* (VCM)[6], dispositivo que atua através de atração e repulsão magnética.

Em 1983, o primeiro HDD com discos de $3\frac{1}{2}$ *in* de diâmetro foi lançado pela empresa Rodime.

Até então, não havia interfaces de disco padronizadas e as novas interfaces eram vendidas junto com os HDs e instaladas em slots *Industry Standard Architecture* (ISA) disponíveis na placa-mãe, (**??**). Em 1985 a Quantum Corporation lançou o *Plus HardCard*, em que tanto a controladora quanto o HD eram integrados em um único slot ISA. Esta integração se mostrou vantajosa pois diminuía problemas de sincronismo e simplificava o projeto, e isto deu origem ao padrão IDE.

Apenas em 1990 o padrão IDE foi ratificado pelo ANSI, organização que tem por objetivo facilitar a padronização dos trabalhos de seus membros, dando origem ao padrão ATA.

Nos anos seguintes, os HDs de $5\frac{1}{4}$ *in* começaram a perder mercado para os HDs de $3\frac{1}{2}$ *in* e $2\frac{1}{2}$ *in*, posteriormente usados no mercado de laptops. Estes formatos foram bem aceitos pelo mercado, sendo utilizados até hoje.

Paralelamente a estes avanços, o desenvolvimento de memórias *Flash* começava a dar os primeiros passos. As memórias *Flash* foram desenvolvidas a partir das memórias EEPROM. Em 1984, o Dr. Masuoka publicou sua invenção no artigo "*A New Flash EEPROM Cell Using Triple Polysilicon Technology* " (**??**).

Em 1987, a Toshiba anuncia as memórias *Flash* do tipo *NAND* e em 1988 a Intel anuncia as memórias *Flash* do tipo *Nor*, (**??**). Hoje, estes dois tipos são considerados os padrões da indústria. A memória *Flash* do tipo *Nand* é otimizada para o armazenamento de dados em massa e a do tipo *Nor* para o armazenamento de dados e código[7]. Em 1988, o primeiro produto *Flash* foi apresentado (**??**), tratava-se de um *chip* de memória *Flash* do tipo *Nor* de 256KB.

Inicialmente, as memórias *Flash* foram usadas para a substituição de memórias EPROM [8] (**??**) e só passaram a ser utilizadas em larga escala no final dos anos

[6]Motores Voice Coil são atuadores cujo funcionamento baseia no de um alto-falante, onde uma bobina é excitada por corrente (sinal de áudio), e um ímã permanente interage com o campo dessa bobina, deslocando um cone e produzindo ondas mecânicas de som.

[7]*Flash* do tipo *Nor* são usadas para armazenar informações na BIOS da placa-mãe e *firmwares* em dispositivos diversos, pois, embora requeiram pouco tempo para a leitura, apresentam um tempo de gravação alto.

[8]EPROM,*Erasable Programmable Read-Only Memory*

1990. Os primeiros HDs SSD baseados em *Flash* só chegaram ao mercado em 2007.

2.3 Padrões Adotados

Com o passar dos anos, várias empresas começaram a fabricar discos rígidos e, visando a compatibilidade entre diferentes discos rígidos e computadores, padrões de tamanho e de interfaces foram adotados.

2.3.1 *Form Factors*

Os discos rígidos foram projetados para serem instalados dentro de computadores, e são produzidos em poucos padrões de tamanho e forma. Estes padrões são chamados de *form factors* de discos rígidos, algo como configurações padrão, e se referem principalmente às dimensões externas dos discos rígidos, (**??**).

A Figura 2.4, mostra seis HDDs com as tampas removidas deixando à mostra os *platters* e as cabeças, onde os padrões de discos de 8, 5.25, 3.5, 2.5, 1.8 e 1 *in* de diâmetro são representados.

Nos HDDs, os *form factors* geralmente são limitados pelo tamanho do disco e os SSDs, embora não tenham esta limitação, geralmente seguem o mesmo padrão para serem compatíveis com as estruturas existentes em *laptops* e *desktops*.

2.3.2 Interfaces

Da mesma forma que outros componentes dos HDs, as interfaces que os interligam passaram por muitas mudanças. Atualmente existem dois tipos predominantes no mercado: ATA e SCSI, ambas são padrões ANSI. Estes padrões definem conectores, conexões físicas, como sinais e temporização, protocolos de transporte, modelos de dispositivos e comandos. Estes modelos e comandos são a base do desenvolvimento do *Framework* proposto neste trabalho e serão detalhados no Capítulo 3.

O padrão ATA, também conhecido como IDE, foi projetado para ser uma interface entre computadores e discos rígidos. Diversas versões foram definidas até agora, são elas: ATA 1 ou IDE, ATA 2 ou *Extended IDE* (EIDE), ATA 3, ATA*Advanced Technology Attachment Packet Interface* (ATAPI) 4, ATA\ATAPI 5, ATA\ATAPI 6, ATA\ATAPI 7 e ATA\ATAPI 8.

A versão ATA 3 foi a primeira a trazer o sistema SMART e, a partir da versão

Figura 2.4: Seis configurações padrão de Discos Rígidos, adaptada de (**??**)

ATA 4, o padrão ATA se tornou capaz de enviar comandos SCSI encapsulados em comandos ATA, possibilitando a comunicação com outros periféricos como unidades de discos ópticos. Esta extensão do protocolo ATA foi chamada de *Advanced Technology Attachment Packet Interface* (ATAPI).

O padrão SCSI foi desenvolvido para ser um padrão genérico para periféricos, como *scanners*, unidades de discos ópticos, mídias removíveis e discos rígidos. Ele foi padronizado pela ANSI em 1986 e dele foram desenvolvidas as seguintes versões: SCSI 1, SCSI 2, com suporte a comandos *multi-thread*, e SCSI 3, com separação entre conexão física, protocolos de transporte e conjuntos de comandos SCSI.

O SCSI oferece melhor desempenho que o ATA em aplicações que requerem velocidade de processamento e transferência de grande quantidades de dados (**??**), por isso os discos SCSI são tipicamente mais caros e, em geral, usados em servidores, onde o alto custo é justificável. Já os discos ATA são mais utilizados em computadores pessoais.

Os computadores se tornaram mais rápidos e as interfaces precisavam atender a esta demanda e, neste intuito, tanto o padrão ATA quanto o SCSI, antes interfaces paralelas, tenderam à serialização, utilizando pares diferenciais, visando alcançar

maiores velocidades de transmissão e evitar problemas de interferência entre as vias. Desta iniciativa surgiram os padrões *Serial ATA* (SATA) e *Serial Attached SCSI* (SAS), protocolos seriais de transporte e interconexão física das interfaces ATA e SCSI, respectivamente, e as interfaces anteriores passaram a ser chamadas de *Parallel ATA* (PATA) e *SCSI Parallel Interface* (SPI).

2.4 *Hard Disk Drive*

De maneira resumida os HDDs são a integração dos seguintes componentes, descritos na Figura 2.5:

- Discos magnéticos (*platters*[9]), onde os dados são armazenados;
- Motor de rotação, responsável por girar os *platters* a uma velocidade angular constante adequada à leitura e gravação de dados;
- Pares de cabeças de leitura e escrita de dados (há duas para cada lado de um *platter*);
- Braço metálico, onde se encontram as cabeças de leitura e escrita;
- Atuador, mecanismo composto por um motor VCM e pelo braço metálico;
- Conector, atualmente a maioria são do tipo SATA ou SAS;
- Placa controladora *Printed Circuit Board* (PCB), que é responsável por acionar o motor que gira os *platters*, posicionar as cabeças de leitura e escrita, através do atuador, e executar os comandos recebidos pelo sistema através de um conector, tais comandos podem ser ATA ou SCSI. A placa controladora pode ser interna, como nos discos SATA, ou externa, como nos discos SCSI, (**??**).

Ao contrário do que se possa pensar, o HDD não é vedado ou blindado. Seu funcionamento se baseia em aerodinâmica. As cabeças não devem entrar em contato com os *platters*, pois isso os danificaria e por isso eles são girados a uma alta velocidade de rotação e só então o braço é movimentado, isto gera uma "bolsa" de ar,

[9] *Platters* são discos magnéticos compostos por duas camadas, o substrato, um disco feito de liga de alumínio ou vidro, extremamente polidos para tentar evitar qualquer rugosidade, e a superfície magnética, que possui apenas alguns micrômetros de espessura (**??**). Neles cada bit é armazenado em um setor de fino segmento destas superfícies através da magnetização do meio pela cabeça de gravação.

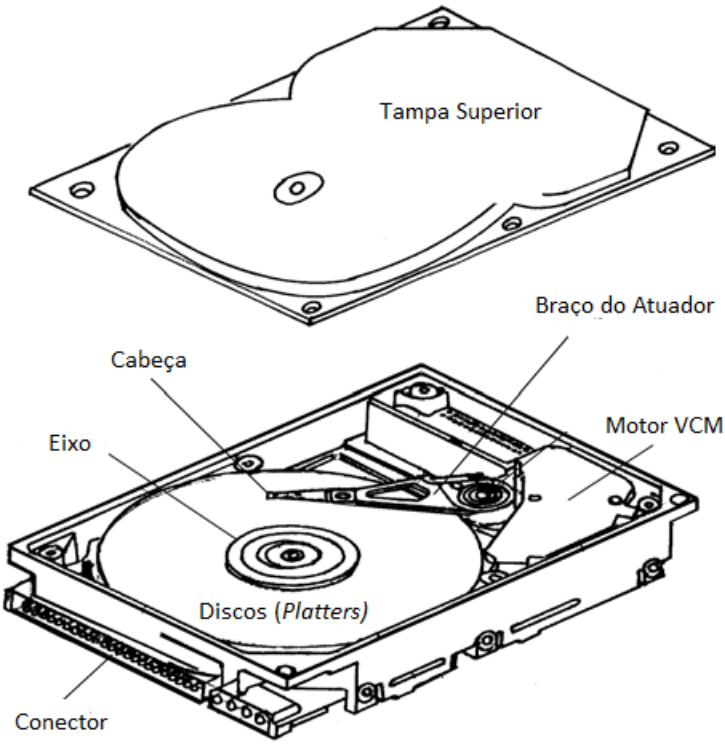


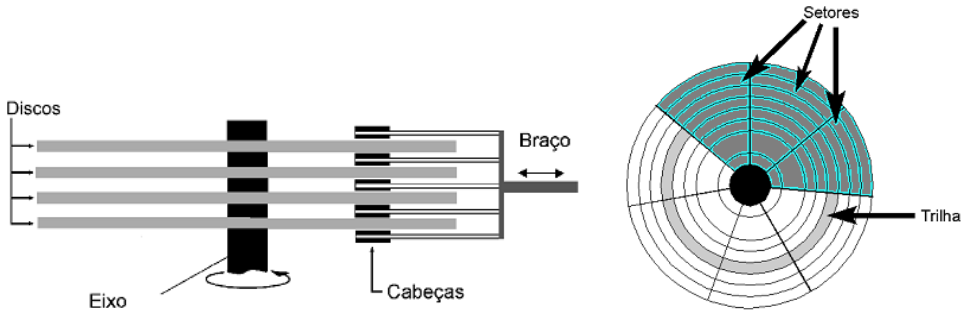
Figura 2.5: Componentes normalmente encontrados em HDDs, adaptada de (**??**)

uma folga entre a cabeça e os *platters*[10], fazendo com que as cabeças flutuem[11] sobre eles. Esta folga pode ser alterada por parâmetros ambientais como a temperatura, a pressão atmosférica e a umidade do ar (**??**).

Nos HDDs existem uma entrada de ar que passa por um filtro, antes de entrar em contato com os *platters*. Embora não seja blindado, abrir um HDD fora de uma sala limpa[12] pode acarretar na inutilização do mesmo, pois qualquer partícula de poeira presente no ambiente poderia danificar a superfície dos *platters* (**??**).

Os HDDs atuais contêm normalmente de 1 a 4 *platters* e, para cada lado de um *platter*, há duas cabeças acopladas, uma de escrita e outra de leitura. Elas estão presas por espaçadores ao braço e embora haja um conjunto de cabeças para cada lado do *platter*, elas não são independentes. Todas se movimentam como uma só, guiadas pelo braço, como mostrado na Figura 2.4.a. Os *platters* são organizados em trilhas concêntricas e estas trilhas organizadas em setores, como mostrado na Figura 2.4.b.

[10]Esta folga é chamada de *Head-Disk Clearance*, (**??**)
[11]Esta bolsa de ar criada funciona como rolamento de ar, o sistema ABS descrito anteriormente
[12]Sala Limpa, ambiente controlado utilizado para testes ou manufatura de produtos onde a contaminação por partículas presentes no ar interfere no resultado



(a) Discos, braço e cabeças de um HDD, adaptada de (**??**)

(b) Trilhas e setores, adaptada de (**??**)

Figura 2.6: Componentes de um HDD

2.5 *Solid-State Drive*

Os Solid-State Drives (SSD) podem ser definidos como dispositivos que armazenam dados em *chips*, na maioria dos casos memórias *Flash*, e sem partes mecânicas, como mostrado na Figura 2.7.

Figura 2.7: Solid-State Drive, adaptada de (**??**)

Os principais componentes de um SSD são a controladora, os módulos de memória *cache*, do tipo *Dynamic RAM* (DRAM), e memória *Flash*, normalmente 10 ou 20, e o conector, que pode ser do tipo SATA, SAS, entre outros (**??**), como mostrado na Figura 2.8.

A controladora, cuja a estrutura é mostrada na Figura 2.9, consiste de um

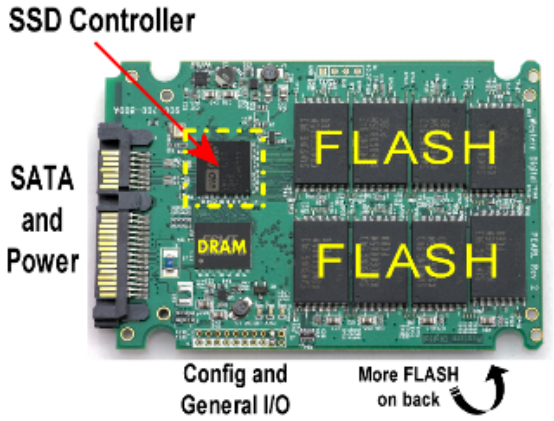


Figura 2.8: Componentes de um SSD, adaptada de (**??**)

processador embarcado, normalmente um microcontrolador de 32 *bits*, que executa código em nível de *firmware*, como o SMART, gerencia automaticamente a memória[13] e executa comandos, como o comando TRIM[14], por exemplo. Ela também gerencia as memórias DRAM, ROM e interfaces de entrada e saída existentes no SSD.

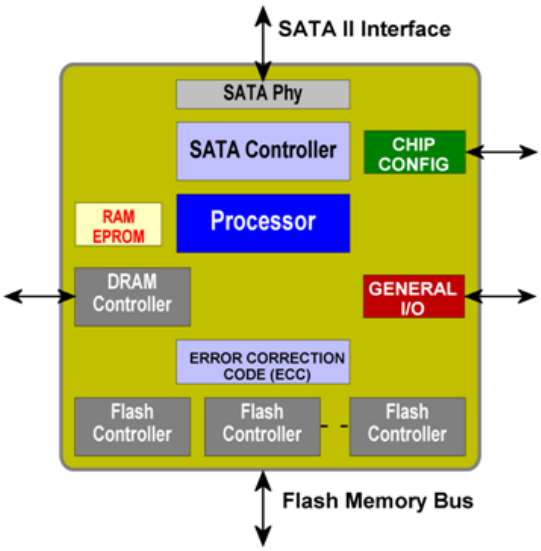


Figura 2.9: Controladora de um SSD, adaptada de (**??**)

Nos SSDs o espaço de armazenamento é dividido em blocos de dados, em geral

[13] *Garbage Collector*

[14] TRIM é um comando ATA para gerenciamento de dados, responsável por informar quais blocos de dados não estão mais em uso,(**??**).

de 512KB, e estes blocos em páginas, usualmente de 4KB. O espaço é preenchido primeiramente sem repetir escritas nos setores, depois que todos os setores são preenchidos.

2.6 *Hard Disk Drive* versus *Solid-State Drive*

Os HDDs e SSDs utilizam as mesmas interfaces, possuem controladora, seja ela interna ou externa, organização dos dados em setores, códigos de correção de erro, endereçamento por *Logical Block Addressing* (LBA)[15] e área reservada de memória (*spare area*). No entanto, as duas tecnologias usam princípios de armazenamento diferentes, e há muitas diferenças de desempenho e de fontes de falha.

2.6.1 Desempenho

Em comparação aos HDDs, os SSDs são menos afetados por choques físicos, pois não contém partes móveis, são mais silenciosos e têm menor consumo de energia elétrica.

Os SSDs têm um tempo de inicialização menor, são mais rápidos na escrita e leitura aleatória e na leitura sequencial de arquivos grandes. Na Figura 2.10, um *benchmark*[16] é mostrado entre HDDs de 7200 rpm e 5400 rpm e um SSD Toshiba MLC NAND[17]. Nele, o dispositivo SSD apresenta resultados bem superiores em termos de taxa de transferência de dados, especialmente na inicialização de aplicações. Entretanto, os *chips* de memória *flash*, usados nos SSDs, tem limites de regravação estimados em 10.000 vezes, se forem do tipo *Multi-Level Cell* (MLC), ou 100.000, se forem do tipo *Single-Level Cell* (SLC) (**??**), o que compromete a sua longevidade, enquanto que nos HDDs, a quantidade de escritas está associada apenas ao desgaste do material ou a danos físicos.

A capacidade de armazenamento e o custo por GB também são pontos que diferem nas duas tecnologias. Até setembro de 2011, o dispositivo SSD de maior capacidade tinha 1.6 TB, *form factor* de 2.5*in* , e o HDD de maior capacidade

[15] Os primeiros HDDs utilizavam endereçamento baseado no espaço físico, o *Cylinder-Head-Sector* (CHS), que identificava um setor pelo cilindro, cabeça e posição na trilha em que estava, mas este modo de endereçamento era limitado pela BIOS e foi substituido pelo LBA, baseado na representação lógica dos setores.

[16] Execução séries de testes e ensaios visando avaliar o desempenho relativo de um produto

[17] O *Benchmark* foi realizado usando processador Intel Core2Duo 2.2GHz E4500, memória cache L2 de 2MB, memória RAM de 2GB DDR SDRAM com 800MHz, placa gráfica Intel GMA3100 e sistema operacional Windows® Vista

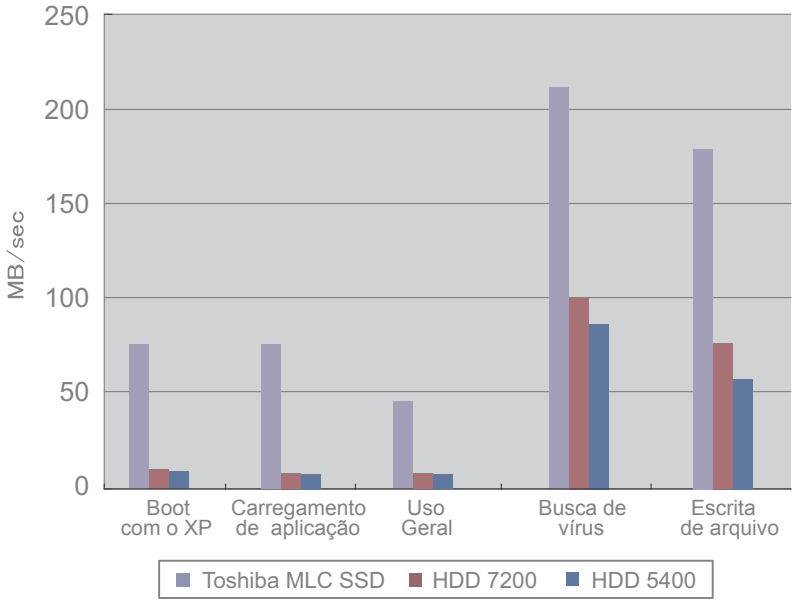


Figura 2.10: *Benchmark* realizado pela PCmark05, adaptada de (**??**)

possuía 4TB, *form factor* de 3.5 *in*. O custo por GB, no dispositivo HDD, era de 0,0625 dólares (**??**) e o do dispositivo SSD, embora não tivesse o preço anunciado, estimava-se um custo por GB superior a 6 dólares, que é custo por GB de um SSD de 1 TB (**??**), o segundo maior dispositivo do tipo. Nestes quesitos, os dispositivos HDD, com custos quase 100 vezes menores, são bastante superiores e sua predominância no mercado é justificada.

2.7 Resumo do Capítulo

Este capítulo mostrou um pouco da história dos discos rígidos, os principais padrões, a evolução das tecnologias SSD e HDD e um comparativo entre elas.

No próximo capitulo serão abordados fundamentos de diagnóstico de falha, o sistema SMART, e as principais características dos padrões ATA e SCSI.

Capítulo 3

Fundamentação Teórica

Neste capítulo são explorados os alicerces deste trabalho: os fundamentos de diagnóstico de falha, os padrões adotados em discos rígidos e como interagir com eles através do Linux.

3.1 Fundamentos de Diagnóstico de Falhas

Primeiramente, para explicar o conceito deste trabalho, é necessário que se defina o conceito de falha, o conceito de diagnóstico e as origens das falhas no dispositivo em estudo, os discos rígidos.

3.1.1 Falha, Erro e Defeito

Embora sejam usadas como sinônimo, estas palavras tem significados diferentes para sistemas de computação. A falha está associada ao universo físico, o erro é a representação da falha no universo da informação, é causado por uma falha, e o defeito é um desvio da especificação e ocorre em consequência de um erro (**??**), as relações de causa entre elas estão ilustradas na Figura 3.1.

Uma falha pode ser ocasionada por um problema em nível de *hardware*, uma flutuação de tensão ou um impacto que cause uma avaria em um setor de um disco rígido, por exemplo, e não necessariamente leva a um estado de erro. O erro acontece quando uma informação é corrompida devido a um setor falho e também não necessariamente leva a um estado de defeito, que é detectado no nível de usuário, pois talvez a informação corrompida nunca seja usada, ou possa ser recuperada.

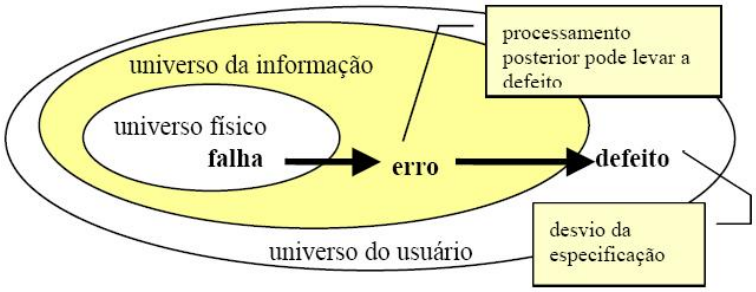


Figura 3.1: Modelo de três universos, adaptada de (**??**)

3.1.2 Diagnóstico de Falhas

Com o propósito de evitar que um sistema se torne defeituoso, as falhas devem ser detectadas e diagnosticadas. A detecção de falhas se baseia em sistemas teóricos, modelos de processo ou funções matemáticas para reconhecer sintomas de falha. Estes sintomas são desvios das características do processo modelado. O diagnóstico de falhas é diferente da detecção de falhas, que consiste em reconhecer que uma falha aconteceu.

Diagnosticar falhas consiste na identificação do tipo de falha com a maior quantidade de detalhes possíveis, bem como a extensão da falha, localização e o momento da detecção (**??**). O procedimento de diagnóstico é baseado na observação e análise de sintomas e no conhecimento prévio sobre o processo. Os sintomas disponíveis podem ser:

Sintomas Analíticos. São o resultado da comparação entre os sinais de um processo e do seu modelo de detecção de falha. Verifica-se se estes sinais excederam algum limiar pré-determinado;

Sintomas Heurísticos. São resultado da análise de um operador humano. Trata-se de impressões quanto a ruído, oscilações, cores e fumaça obtidos por inspeção. Tratados muitas vezes como medidas qualitativas;

Histórico de processos e estatísticas de falha. Este histórico inclui informações sobre tempo de execução e reparo e descrevem a frequência de certas falhas no processo;

Representação unificada dos sintomas. É a observação conjunta de sintomas analíticos e heurísticos, através de mecanismos de inferência de sistemas *fuzzy*;

Relações Falha-Sintomas. Consistem na análise inversa das relações de causa e efeito. Na análise direta, observam-se falhas que geram eventos que geram sintomas. Na análise inversa, as falhas são deduzidas a partir dos sintomas observados;

3.1.3 Falhas em Discos Rígidos

Segundo (**??**), as falhas em discos rígidos podem ser ocasionadas por falhas no meio de armazenamento, que podem causar a perda de dados, ou por outras falhas como problemas de alimentação, problemas na controladora do disco ou software, por exemplo, e que não necessariamente causam a perda de dados, mas levam à sua indisponibilidade temporária.

Os discos rígidos são organizados em pequenos módulos de armazenamento, chamados de setores, nos HDDs, ou blocos, nos SSDs. Aqui estes módulos serão chamados genericamente de setores. Estes setores dispõem das seguintes informações: identificação, usada para localizar o setor, como LBA, por exemplo; dados, a informação em si; campos de sincronização, usados internamente pela controladora para guiar o processo de leitura; espaçadores (*Gaps*), usados para separar setores e dar tempo para que a controladora processe o que estava sendo lido; e códigos de correção de erro, como *Cyclic Redundancy Check* (CRC), por exemplo (**??**), que são usados pela controladora para detectar a ocorrência de falhas.

As falhas em discos rígidos podem ser divididas quanto à duração, como temporárias ou permanente; e quanto à magnitude, como parcial, intermediária ou total (**??**). Falhas temporárias podem ser causadas por mudanças de estado do sistema. Um exemplo de situação de falha temporária é quando um comando não é completado com sucesso e através da repetição ele pode ser completado sem erros. Falhas permanentes são duradouras, mas os erros causados por elas talvez possam ser contornados por códigos de correção de erro ou leitura de outros setores, por exemplo. Em relação à magnitude, uma falha pode afetar todo o disco ou uma pequena parte dele. Normalmente, um setor é a mínima área afetada.

As falhas podem ser geradas no processo de fabricação. Alguns HDs novos já saem de fábrica com setores falhos[1] e outros setores se danificam em decorrência do uso e desgaste natural. Os setores falhos são remapeados em uma área reservada. Enquanto que nos HDDs isto pode gerar perda de desempenho, pois gera mais

[1] Em inglês *Bad blocks* ou *Bad sectors*

movimentos de busca para ler um arquivo, por exemplo, nos SSDs este mapeamento é praticamente transparente. A área reservada tem duas funções principais: servir de armazenamento temporário de novos dados e ser uma espécie de memória "reserva" , onde os setores com falha são remapeados, aumentando a longevidade do disco rígido (**??**).

A área reservada é gerenciada pela controladora e se relaciona com a diferença entre a nomenclatura decimal utilizada pelos fabricantes e a nomenclatura binária usada pelos sistema operacional. Por exemplo, cada 1GB de memória na nomenclatura do fabricante corresponde a 10^9 *bytes*, o que para um sistema operacional corresponde a 2^{30} *bytes* e a diferença entre as duas notações, igual a 73.741.824 *bytes*, é a quantidade de memória a ser usada como área reservada.

O remapeamento de setores com falha é transparente para o sistema operacional. Quando um setor falho é requisitado, a controladora intercepta e devolve o dado correto do novo bloco atribuído àquele endereço na área reservada. Apenas quando esta área está totalmente cheia, o sistema operacional começa a indicar a existência de setores falhos.

Nos SSDs os tipos mais comuns de falha em células de memória estão relacionados a distúrbios de gravação, problemas na retenção de dados e defeitos de fabricação. Cada célula armazena um bit de memória. Os distúrbios de gravação consistem na corrupção de dados de células escritas durante a escrita de outras células na mesma matriz de memória (**??**). Estes distúrbios podem acontecer nas células de mesma coluna ou linha.

Nos HDDs, assim como nos SSDs, existem distúrbios de gravação, problemas na retenção de dados e defeitos de fabricação. Os distúrbios de gravação podem ocorrer entre setores adjacentes, trilhas vizinhas e até mesmo entre *platters*, já que há um movimento unificado à todas as cabeças. Um exemplo de distúrbio de gravação é a propagação radial do erro (**??**), onde, por algum motivo, uma trilha não é escrita como um círculo perfeito[2] e, quando esta é referenciada para leitura, a cabeça segue um círculo perfeito assim como para as trilhas seguintes, onde os setores desejados não estão e outros setores são lidos.

Além das falhas originadas pelo desgastes ou defeitos de fabricação do meio magnético, os HDDs podem apresentar várias fontes de falhas inseridas pelas partes

[2]Este fenômeno é chamado *track misregistration* (TMR)

móveis assim como por fatores ambientais e impactos físicos. Por exemplo, quando uma tentativa de leitura de um setor falha, a causa tanto pode ser um distúrbio de gravação, quanto uma falha no atuador, que não conseguiu posicionar a cabeça corretamente. Outro exemplo é a folga existente entre as cabeças e os *platters*, que diminui com o aumento da temperatura, com a diminuição da pressão atmosférica, e também com o aumento da humidade do ar. Esta diminuição da folga pode levar ao choque das cabeças com pontos de maior rugosidade nos *platters* e estes choques podem gerar erros de leitura e gravação e danificar o disco rígido.

Há dois meios de detectar setores falhos em discos rígidos: através de requisições normais do sistema operacional, leituras e escritas, e através de buscas, como execuções de algoritmos de testes, por exemplo. Segundo (**??**), as falhas em discos rígidos estão fortemente relacionadas umas com as outras, ou seja, a probabilidade de um setor falhar é significativamente maior em áreas próximas às regiões de falhas conhecidas. Falhas temporárias são o primeiro sinal de deterioração do meio e podem ser usadas para predizer falhas permanentes. Estas informações podem ser utilizadas no desenvolvimento de algoritmos de teste.

3.2 SMART

A indústria de HDs adotou um sistema de monitoramento para discos rígidos, o sistema SMART, visando detectar falhas, disponibilizar diferentes tipo de auto-teste e relatar medidas indicadoras de confiabilidade. O SMART prevê falhas em discos rígidos individualmente, diferentemente dos modelos de confiabilidade, que depois de análises estatísticas calculam a probabilidade de falha de uma população de discos rígidos, supondo que todos são iguais. O SMART prevê falhas através de medidas e contagens, como a quantidade de setores realocados, taxa de erro de leitura, a quantidade de choques físicos sofridos e o tempo total de uso, por exemplo. Na Tabela 3.2 alguns atributos monitorados pelo SMART são mostrados.

Um algoritmo de alerta de falha é executado pelo *firmware* da controladora do disco rígido e checa se as medidas excederam o limiar máximo e gera um alerta binário do tipo: Irá falhar, Não Irá Falhar. Os limiares máximos são definidos pelos fabricantes. O objetivo é gerar o alerta 24 horas antes de o *drive* falhar, (**??**).

Há comandos padronizados para habilitação e leitura desses alertas de falha pelos sistemas operacionais. Alguns computadores checam o SMART durante a inicialização do sistema operacional e alguns fabricantes disponibilizam programas

Tabela 3.1: Exemplos de atributos SMART para ATA

Identificador	Atributo	Descrição
01	Taxa de erro de leitura (*Read Error Rate*)	Frequência de erros durante a leitura de dados do disco.
02	Desempenho (*Throughput Performance*)	Eficiência média do disco.
05	Contagem de setores realocados (*Reallocated Sectors Count*)	Contagem dos setores realocados na área reservada.
08	Desempenho das operações de busca (*Seek Time Performance*)	Desempenho médio das operações de busca nos discos magnéticos. A diminuição deste atributo pode indicar problemas no sistema mecânico.
191	Taxa de erro devido a impactos (*G-sense Error Rate*)	Frequência de erros resultantes de impactos e vibrações.
194	Temperatura (*Temperature*)	Temperatura interna do disco.
197	Contagem atual de setores pendentes (*Current Pending Sector Count*)	Número de setores esperando para serem remapeados.

de diagnóstico que lêem os alertas do SMART (**??**). O sistema SMART foi especificado para o padrão ATA, detalhado na próxima seção e o seu equivalente em discos SCSI é a combinação dos comando SEND DIAGNOSTIC e RECEIVE DIAGNOSTIC RESULTS. O SMART também disponibiliza testes para verificar as condições do disco rígido (**??**), (**??**). Estes testes são descritos a seguir.

SMART *Short Self-Test.* Avalia o desempenho elétrico, mecânico e de leitura do disco. Realiza buscas em pequenas áreas do disco e checa a lista de setores pendentes (setores com falhas e que ainda não foram remapeados)[3], com prováveis erros de leitura.

SMART *Extended Self-Test.* Segue o mesmo princípio do SMART *Short Self-Test*, mas realiza a leitura de toda a superfície do disco, geralmente é um processo bastante demorado.

SMART *Conveyance Self-Test.* identifica danos ocorridos durante o transporte

[3]A controladora armazena informações como esta em *logs*, que podem ser acessados por comandos específicos, ATA ou SCSI

do disco rígido.

SMART *Selective Self-Test.* executa teste apenas em alguns trechos do disco.

3.3 Protocolos

Como dito anteriormente, ATA e SCSI são os padrões de comunicação utilizados para discos rígidos. Eles são planejados em camadas. São elas: camada de aplicação, onde são definidos conjuntos de comandos e especificações, camada de transporte, onde são definidos protocolos de transporte e serviços, e camada de interconexão física, que definem conectores, cabeamento, mecanismos de sinalização (**??**) (**??**).

3.3.1 SCSI

O SCSI é um padrão genérico que permite conectar computadores a discos rígidos, *scanners*, leitores ópticos, entre outros dispositivos. A conexão SCSI pode acontecer entre quaisquer dois dispositivos SCSI. Um deles é o *iniciador*, que começa a transmissão, e o outro é o *alvo*, que recebe os comandos e os executa (**??**).

A Figura 3.2 mostra o caminho percorrido pelos comandos enviados pelo dispositivo **iniciador** até chegar ao dispositivo **alvo**. O **iniciador** envia um comando, que é tratado por um protocolo de transporte, de onde é transmitido para o dispositivo **alvo** através de uma conexão física. Ao chegar no dispositivo **alvo** o comando passa por um protocolo de transporte e é então processado e executado por ele.

Note que os protocolos de transporte e interconexões físicas não interferem no comando enviado. Logo, outros padrões podem ser utilizados para transmitir comandos SCSI, como acontece com o padrão ATAPI, onde comandos SCSI são encapsulados e mandados através do SATA ou PATA, protocolos de transporte e de interconexão física ATA.

Na Figura 3.3 a família de padrões SCSI é apresentada (**??**). No topo da figura estão o conjunto de comandos primários[4], compartilhado por todos os tipos de dispositivos, e os conjuntos de comandos específicos, utilizados por diferentes grupos de dispositivo, como os de comandos de blocos[5] (caso dos HDs), de *stream*[6], de

[4] *SCSI Primary Commands - 4 (SPC-4)*, versão mais recente.
[5] *SCSI Block Commands - 3 (SBC-3)*, versão mais recente.
[6] *SCSI Stream Commands - 4 (SSC-4)*, versão mais recente.

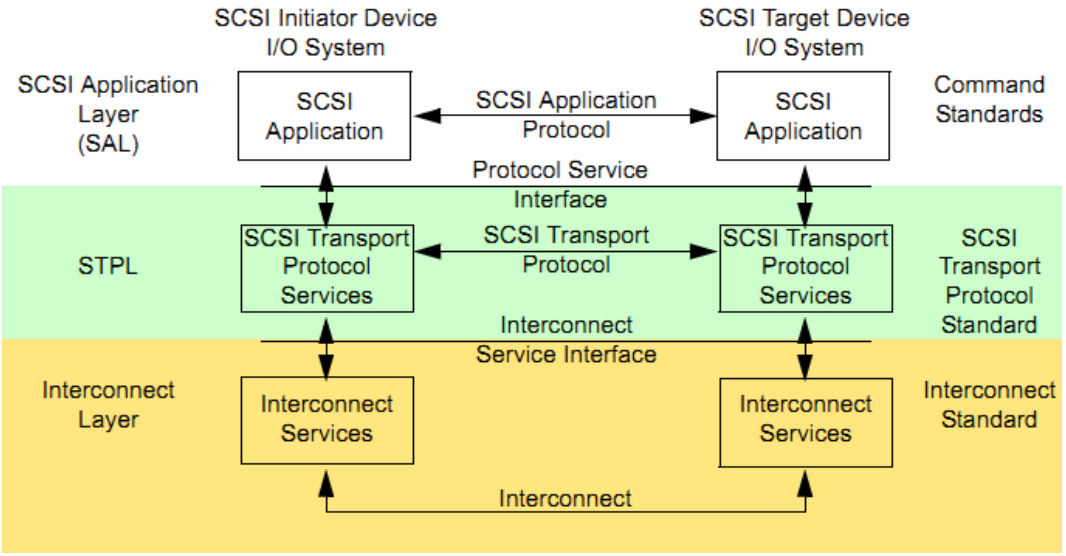


Figura 3.2: SCSI, modelo de camadas, adaptada de (**??**)

multimídia[7], ou da controladora[8], por exemplo.

No meio da figura estão os protocolos de transporte, SAS[9], *fibre channel*[10], *Internet SCSI* (iSCSI), entre outros. Na base estão as conexões físicas, SAS[11], *fibre channel*, internet, *Universal Serial Bus* (USB), *PCI express*, entre outros. Já na lateral da figura está o modelo de arquitetura SCSI[12].

3.3.2 ATA

O padrão ATA foi desenvolvido para ser uma interface entre computadores e discos rígidos. Inicialmente projetado para conectar placas-mãe e discos rígidos com as novas versões passou a conectar dispositivos ópticos, como leitoras de CD, DVD, e BD(*BluRay Disc*), e também fitas magnéticas, através do padrão ATAPI, que permite a transmissão de comandos SCSI por conexões ATA.

O ATA permite comunicar um *host*, que pode ser um *software*, o BIOS ou um *driver* de um sistema operacional, a um dispositivo, que pode ser um disco rígido ou um leitor óptico, através de um subsistema de entrega. A Figura 3.4 mostra como essa comunicação é realizada. O *host* envia um comando, que passa por um

[7] *MultiMedia Command Set - 6 (MMC-6)*, versão mais recente.
[8] *SCSI Controller Commands-2 (SCC-2)*, versão mais recente.
[9] *SAS Protocol Layer - 3 (SPL-3)*
[10] *Fibre Channel Protocol - 4 (FCP-4)*, versão mais recente.
[11] *Serial Attached SCSI - 3 (SAS-3)*, versão mais recente.
[12] *SCSI Architecture Model - 5 (SAM-5)*, versão mais recente.

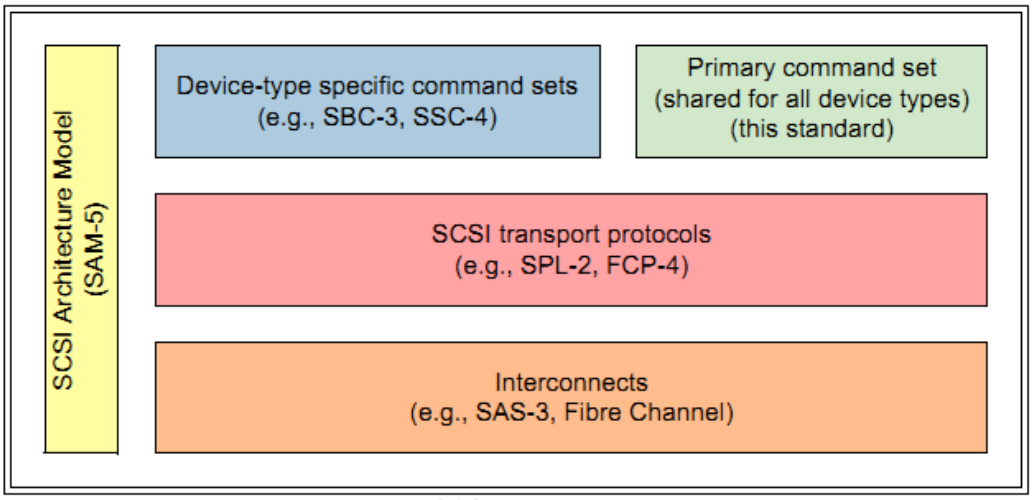


Figura 3.3: Família de padrões SCSI, adaptada de (**??**)

protocolo de transporte, de onde é transmitido para o dispositivo através de uma conexão física. Ao chegar no dispositivo, o comando passa por um protocolo de transporte e é então processado e executado por ele.

Assim como no padrão SCSI, os protocolos de transporte e interconexões físicas não interferem no comando enviado. Logo, outros padrões podem ser utilizados para transmitir comandos ATA, como acontece com o padrão SCSI, que pode transmitir comandos ATA através de uma camada de tradução[13]. Comandos ATA são encapsulados em comandos ATA Pass-Through e mandados através do SATA ou PATA, protocolos de transporte e de interconexão física SCSI.

O desenvolvimento do ATA é guiado por um modelo abstrato, especificado através de diversos documentos, também chamados de padrões, como descrito na Figura 3.5, adaptada de (**??**). Nela tem-se o modelo de arquitetura[14], que define o modelo de sistema e especificações, o conjunto de comandos[15], o conjunto de comandos de entrega, que são documentos SCSI e ATAPI, e os padrões de transporte PATA e SATA.

[13] *SCSI-ATA Translation*
[14] *ATA/ATAPI Architecture Model (ATA8-AAM)*, versão mais recente.
[15] *ATA/ATAPI Command Set (ATA8-ACS)*, versão mais recente.

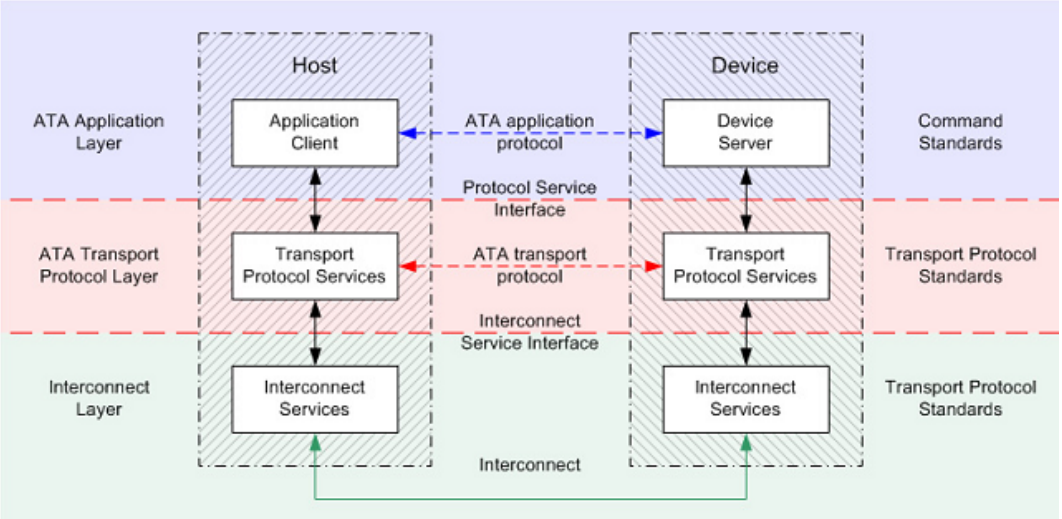


Figura 3.4: ATA, modelo de camadas, adaptada de (**??**)

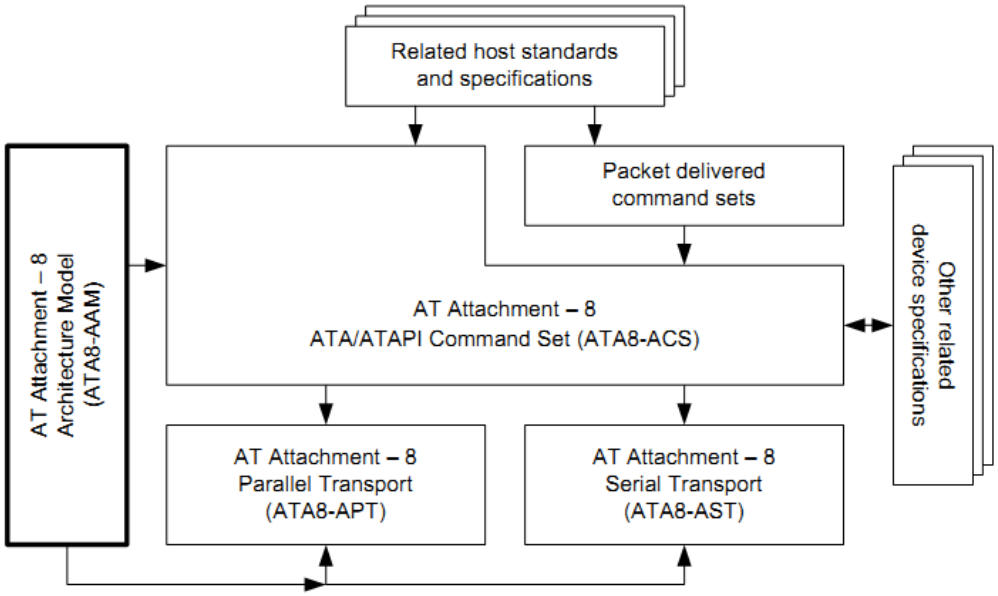


Figura 3.5: Família de padrões ATA

3.4 Resumo do Capítulo

Neste capitulo foram explorados os conceitos de falha, erro e defeito, o diagnóstico de falhas, bem como os tipos de falha que ocorrem em discos rígidos. O sistema SMART e os protocolos ATA e SCSI, que são a base de desenvolvimento deste trabalho, foram apresentados.

No próximo capítulo a metodologia utilizada neste trabalho e o desenvolvimento deste trabalho são apresentados.

Capítulo 4

Metodologia e Implementação

Este capítulo apresenta as metodologias de desenvolvimento e de teste utilizadas neste projeto, bem como os detalhes de implementação do *framework* proposto e os algoritmos implementados. Na Seção 4.2 as estratégias de implementação dos comandos ATA e SCSI são apresentadas, as abordagens adotadas são discutidas e é feito o detalhamento do *framework*, da camada de aplicação e do *Bootable* criado.

A Seção 4.1 descreve o ambiente de desenvolvimento e os recursos utilizados, comenta o método de inserção de falhas utilizado e descreve a metodologia de teste.

4.1 Metodologia

No desenvolvimento deste trabalho, foram utilizados diversos HDs, para validação dos comandos e algoritmos, *laptops* e *desktops*, para realização dos teste e validação do funcionamento do *Bootable*. Todos os computadores e HDs utilizados, exceto o computador pessoal da autora, fazem parte do acervo do Laboratório de Engenharia de Sistemas de Computação (LESC)[1], da Universidade Federal do Ceará (UFC).

Na validação dos comandos foram gerados pequenos módulos executáveis para que os comandos pudessem ser validados individualmente. Foi desenvolvido um método de inserção de falha descrito na Seção 4.1.1, usado para validar os comandos e os algoritmos de teste.

[1]Laboratório de pesquisa e desenvolvimento com vasta experiência no desenvolvimento de *softwares* de diagnóstico para linhas de produção e usuários finais.

4.1.1 Inserção de Falhas

Na validação dos algoritmos foram utilizados HDs em boas condições, sem falhas detectadas por outras ferramentas de diagnóstico, nos quais foram "inseridos" falhas em alguns setores, através do comando ATA WRITE UNCORRECTABLE (45H). Com este comando é possível configurar setores para que retornem *status* de falha, quando um comando de leitura for executado. Há duas opções disponíveis, criar um erro pseudo incorrigível, com *log*, e criar um erro sinalizado, sem *log*. Na primeira opção, quando um comando de leitura for executado, o erro de leitura será incluído nos *logs* da controladora e do SMART e na segunda opção o comando retornará erro, mas este não será inserido nos *logs*. Neste trabalho utilizou-se a primeira opção, para ajudar na validação de comandos, como READ LOG EXT (2FH), e na implementação de testes, como o *Targeted Read*, que se baseia na leitura dos *logs* armazenados.

Para remover a falha inserida é necessário realizar um comando de escrita no setor configurado, o dado armazenado será perdido e o setor não retornará *status* de erro na realização de novos comandos de leitura, a menos, é claro, que este setor se torne realmente falho. Neste trabalho utilizou-se os comandos WRITE SECTOR(S) (30H) e WRITE SECTOR(S) EXT (34H) para remover as falhas "inseridas".

4.1.2 Metodologia de Teste

Os testes foram realizados utilizando discos rígidos com falhas conhecidas, detectadas por outros *softwares*, no caso Lenovo *ThinkVantage ToolBox*, desenvolvido pela PC-Doctor disponível nos computadores Lenovo e executado no Windows, e *smartmootools*, executado no Linux.

Os algoritmos implementados foram testados utilizando 12 HDs nos quais estes algoritmos foram executados. Destes HDs 9 são do tipo HDD e 3 do tipo SSD, entre eles há HDs com falhas e HDs em perfeito estado de funcionamento. Foram utilizados discos de dois tipos de *form factor*, $2.5in$ e $3.5in$. Uma sequência de algoritmos foi definida e executada 3 vezes para cada dispositivo. Os resultados foram comparados e estão descritos no Capítulo 5.

4.2 Implementação

O intuito deste trabalho é o desenvolvimento de um *framework* para dar suporte ao desenvolvimento de testes de diagnóstico para serem executados em *desktops* e

laptops. Neste trabalho alguns testes foram implementados, apenas para demonstrar a eficiência do *framework* e servir de base para uma análise preliminar sobre a validade dos mesmos, pois, tendo em vista que para uma análise estatística confiável destes algoritmos seria necessária uma grande quantidade de discos rígidos, variações de parâmetros e repetições de testes, e não seria possível concluir no tempo previsto, de um semestre.

Como dito anteriormente, a capacidade de armazenamento dos discos rígidos vem crescendo fortemente. Testar todos os setores de um disco rígido pode levar muitas horas e *softwares* de diagnóstico como PC-Doctor, Aida32 e Hard Disk Sentinel, oferecem testes rápidos, que analisam amostras de setores, que levam de 5 a 10 minutos, e é para o desenvolvimento deste tipo de teste que o *framework* foi pensado.

Na Seção 4.2.1 a implementação dos comandos ATA e SCSI no Linux é descrita. O *framework* proposto é descrito na Seção 4.2.2 e o *bootable* na Seção 4.2.4.

4.2.1 ATA, SCSI, SMART e o Linux

No Linux, lidar com dispositivos SCSI e ATA é bastante simples, similar ao trato de arquivos de entrada e saída padrão. Os comandos são enviados aos dispositivos através dos *drivers*, *softwares* que conectam o sistema operacional ao *hardware* do computador (**??**).

Os dispositivos correspondem a arquivos no diretório */dev* e utilizam-se de chamadas de sistema, por exemplo, para comandá-los. Usa-se a chamada *open()* para criar um descritor para o dispositivo, como mostrado no próximo parágrafo. Uma vez criado o descritor, os comandos podem ser escritos e lidos através dele com as chamadas *read()*, *write()* ou *ioctl()*, como descrito em (**??**), (**??**).

```
int fd = open (device_name, O_RDWR);
```

Discos SCSI são acessados através do *driver sd*, para discos rígidos, e *sg*, para quaisquer dispositivos SCSI, ambos disponibilizados pelo pacote sg3-utils. Neste trabalho, os dados serão escritos e lidos usando a estrutura sg_io_hdr_t, descrita em (*scsi/sg.h*) e a seguir, enviadas através do *driver sg*.

```
int dxfer_direction: especifica a direção da transferência de dados.
unsigned char *dxferp: ponteiro para os dados transferidos.
```

Bit Byte	7	6	5	4	3	2	1	0
0	OPERATION CODE							
1	Miscellaneous CDB information			SERVICE ACTION (if required)				
2	(MSB)	LOGICAL BLOCK ADDRESS (if required)						
3								
4								
5								(LSB)
6	Miscellaneous CDB information							
7	(MSB)	TRANSFER LENGTH (if required) PARAMETER LIST LENGTH (if required) ALLOCATION LENGTH (if required)						
8								(LSB)
9	CONTROL							

Figura 4.1: Comando de 10 bytes típico, adaptado de (**??**)

unsigned char * cmdp: ponteiro para o comando.
unsigned char * sbp: ponteiro para o sense buffer.
unsigned char masked_status: status GOOD ou CHECK CONDITION.

Os comandos SCSI têm, em geral, 6, 10, 12 ou 16 *bytes* de tamanho e são enviados no formato de um bloco descritor de comando, que é um dos campos da estrutura sg_io_hdr_t e contém *opcode* do comando, LBA (endereço do setor), tamanho em *bytes* a ser transferido e outras flags de comando, como descrito na Figura 4.1, um comando de 10 bytes.

Depois de receber um comando, o dispositivo *alvo* responde com um *byte* de *status* e retorna uma estrutura chamada sense buffer com mais informações. Utilizam-se algumas abordagens mostradas em (**??**), (**??**), quanto ao uso das chamadas *read()/write()* ou *ioctl*. Neste trabalho foi adotado o uso da chamada *ioctl*, que equivale ao uso de uma chamada *write()* seguida de uma chamada *read()*. No caso de um comando que retorne erro, usando *read()/write()*, é necessário o envio de outro comando, o REQUEST SENSE, para obter mais informações sobre o problema; com o uso do *ioctl()* a própria estrutura já retorna todas as informações, no campo *sense_buffer*. No próximo parágrafo o envio de um comando por *ioctl()* é descrito, onde fd é o descritor do dispositivo, p_hdr é uma instanciação da estrutura sg_io_hdr_t e SG_IO é o parâmetro referente ao *driver sg*.

```
int ret = ioctl(fd, SG_IO, p_hdr);
```

O processamento em discos SATA/ATA é um pouco diferente, pois algumas vezes, as controladoras SATA estão conectadas através de um barramento do tipo PCI, o que leva o sistema operacional a interpretá-lo como um adaptador de barramento[2] SCSI (**??**). Isto impossibilita o envio de comandos ATA aos discos, para ter acesso aos discos utiliza-se a camada de tradução SCSI-ATA, que encapsula comandos ATA em comandos SCSI, provida no Linux pela biblioteca libATA (**??**), fazendo uso do comando ATA Pass-Through. A camada de tradução é descrita em (**??**) e o comando é detalhado em (**??**). Como o sistema SMART é especificado pelo padrão ATA, os comandos de SMART também são enviados via ATA Pass-Through.

O envio de comandos via ATA Pass-Through apresentou alguns detalhes peculiares durante o desenvolvimento. A ordem de preenchimento da estrutura de dados do comando a ser enviado influencia no resultado do comando. Na Figura 4.2 a estrutura do comando é apresentada. Diversos testes foram realizados em diferentes discos rígidos e verificou-se que apenas quando os campos *LBA_LOW(0:7)*, *LBA_MID(0:7)*, *LBA_HIGH(0:7)* eram preenchidos em sequência o comando respondia adequadamente. Não foram encontradas referências a isso na documentação dos padrões ATA e SCSI, apenas uma sequência de bytes comentados em um código fonte que implementava um dos comandos SCSI. Outro detalhe é que a resposta dos comandos enviados via ATA Pass-Through varia de acordo com o fabricante, pois uma certa quantidade de *bytes* é coletada antes da estrutura que descreve a resposta. Esta quantidade varia de acordo com o fabricante, em torno de 22 *bytes*, depois destes *bytes* a sequência de *bytes* 09h 0ch marca o início da estrutura de descrição de resultados, também não há referência a isto nas documentações dos padrões ATA e SCSI.

4.2.2 *Framework* Proposto

O *framework* foi implementado em linguagem C++, escolhida por reunir características de linguagens de alto e baixo nível. Esta foi desenvolvida para ser compatível e tão eficiente e portável quanto a linguagem C e suporta múltiplos paradigmas de programação, principalmente a programação estruturada e a programação orientada a objetos.

Os padrões ATA e SCSI disponibilizam vários tipos de comandos, desde

[2] *Host Bus Adapter (HBA).*

Byte\Bit	7	6	5	4	3	2	1	0
0	OPERATION CODE (85h)							
1	MULTIPLE_COUNT			PROTOCOL				EXTEND
2	OFF_LINE		CK_COND	Reserved	T_DIR	BYT_BLOK	T_LENGTH	
3	FEATURES (8:15)							
4	FEATURES (0:7)							
5	Reserved / SECTOR_COUNT (8:15)							
6	SECTOR_COUNT (0:7)							
7	Reserved / LBA_LOW (8:15)							
8	LBA_LOW (0:7)							
9	Reserved / LBA_MID (8:15)							
10	LBA_MID (0:7)							
11	Reserved / LBA_HIGH (8:15)							
12	LBA_HIGH (0:7)							
13	DEVICE							
14	COMMAND							
15	CONTROL							

Figura 4.2: Comando ATA Pass-Through, adaptado de (**??**)

comandos simples para a leitura de informações sobre o fabricante do dispositivo até comandos para aplicações de segurança, e nem todos os comandos são úteis para este trabalho, que se propõe a execução de algoritmos de leitura e testes SMART. Alguns comandos foram selecionados e são listados nesta Seção. Foram implementados os principais comandos de leitura e verificação de LBAs, comandos SMART e de auto-teste e comandos para leitura dos *logs* gerados pela controladora. Estes *logs* contêm informações como os comandos que retornaram erro e em que LBAs eles foram executados, como descrito na Seção 3.1.3, isto é um recurso útil para a verificação de falhas temporárias e checagem da vizinhança.

O *framework* é descrito como um diagrama de classes na Figura 4.3. Nesta Figura, a classe *ScsiCmd* implementa a estrutura de um comando SCSI, bem como as suas funcionalidades, várias classes são herdeiras da classe *ScsiCmd*, entre elas a classe *ATAPassThrough*, de onde os comandos ATA, entre eles o SMART, herdam suas características.

Neste diagrama são descritas apenas as classes que implementam comandos ATA e SCSI e que foram utilizadas na implementação de testes. Todos os comandos implementados são detalhados nas Tabelas 4.1 e 4.2. As classes *Inquiry*, *ModeSense*, *ReadCapacity*, *Read*, *Verify* e *TestUnitRead* implementam comandos SCSI puros e

podem ser usados para outros tipos de dispositivos. No centro do diagrama está a classe *AtaPassThrough*, dela são herdeiras as classes *ReadVerify*, *ReadVerifyExt*[3], *ReadSector*, *ReadSectorExt*, *Identify*, *ReadLogExt*, *WriteSector*, *WriteUncorrectable* e *SmartCmd*, estes comandos são ATA. As classes ligadas à *SmartCmd* implementam um mesmo comando, que contém várias funcionalidades diferentes. São elas: *SmartStatus*, *EnableSmart*, *DisableSmart*, *SmartReadLog*, *AbortSelfTest*, *ShortSelfTestSmart*, *ExtendedSelfTestSmart* e *ConveyanceSelfTestSmart*.

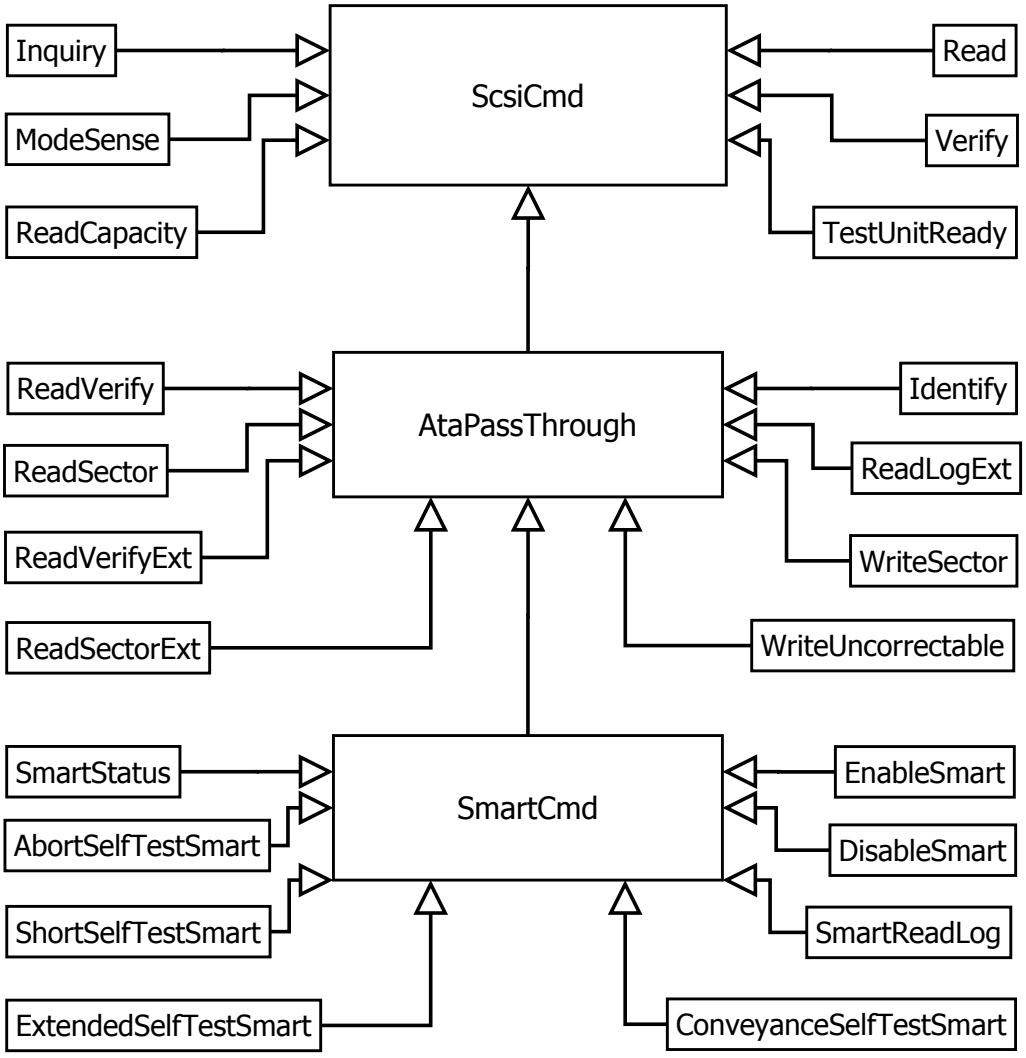


Figura 4.3: Diagrama de Classes do Framework

A Tabela 4.1, descreve os comandos SCSI implementados e a Tabela 4.2 os

[3]A diferença entre *ReadVerify* e *ReadVerifyExt*, assim como *ReadSector* e *ReadSectorExt*, é o tamanho da acLBA endereçada. Os comandos "Ext" utilizam LBA de 48 bits e os outros de 28 bits.

comandos ATA, enviados via ATA Pass-Through.

4.2.3 Camada da Aplicação

Na camada de aplicação os algoritmos de teste são implementados e também outras funções para auxiliar na execução do teste como funções de listagem de dispositivos, filtragem de *pendrives* e discos rígidos externos, tratamento de *strings* e uso de expressão regular, que não serão detalhados aqui por não ser o foco deste trabalho.

Os testes são voltados para computadores em uso. Logo, eles não podem realizar escritas no disco rígido, pois isso poderia corromper os dados ou o sistema de arquivos do usuário. Os dispositivos SSDs tem comportamento físico "similar" a memórias RAM, para as quais existem vários testes consagrados como os testes do tipo *MARCH*, *Moving Inversion*, *GALPAK* (**??**). Entretanto, todos estes algoritmos utilizam operações de escritas na verificação da memória, o que impossibilita o seu uso para diagnóstico de discos rígidos.

Os algoritmos implementados são de dois tipos, SMART e de busca, que se baseiam na descrição dada nos manuais de ajuda pelos *softwares* de diagnóstico, pois como se trata de uma linha de estudo bastante comercial, há pouca ou nenhuma informação sobre eles. Os algoritmos são descritos a seguir:

- SMART *Return Status* executa o comando SMART *Return Status* e verifica se as medidas coletadas excedem o limiar máximo. Este comando retorna um alerta binário do tipo: Irá Falhar/Não Irá Falhar. Este alerta pretende avisar ao usuário quanto à probabilidade de o sistema falhar durante as próximas 24 horas de funcionamento.

- SMART *Short Self-Test* executa o comando SMART *Execute off-line immediate*, com o subcomando *Execute SMART short self-test routine in off-line mode*.

- SMART *Extended Self-Test* executa o comando SMART *Execute off-line immediate*, com o subcomando *Execute SMART extended self-test routine in off-line mode*.

- SMART *Conveyance Self-Test* executa o comando SMART *Execute off-line immediate*, com o subcomando *Execute SMART conveyance self-test routine*

Tabela 4.1: Comandos SCSI selecionados em ordem alfabética

Comando	Código	Descrição
ATA Pass-Through (16)	85h	Este comando permite o envio encapsulado de comandos ATA através de comandos SCSI
Inquiry	12h	Permite coletar informações sobre o dispositivo, o fabricante entre outros.
Log Select	4Ch	Permite coletar informações dos *logs* da controladora.
Log Sense	4Dh	Permite coletar informações dos *logs* da controladora. Funciona em conjunto com o comando Log Select
Mode Sense (6)	1Ah	Permite coletar informações de páginas específicas do dispositivo.
Read (10)	28h	Permite a leitura de um setor específico do dispositivo. Requisita a transferência de dados.
Read Capacity (10)	25h	Permite coletar informações sobre a capacidade do dispositivo, devolve a última LBA válida.
Read Defect Data (10)	37h	Permite coletar informações sobre os setores defeituosos no dispositivo.* [4]
Reassign Blocks	07h	Envia a requisição para que a controladora remapei os setores defeituosos.*
Receive Diagnostic Results	1Ch	Permite coletar informações da página específica de diagnóstico. Usado em conjunto com Send Diagnostic.
Send Diagnostic	1Dh	Requisita a execução de auto-testes pelo dispositivo.
Test Unit Ready	00h	Testa se a unidade está pronta para receber comandos
Verify (10)	2Fh	Realiza a verificação de um setor, compara o conteúdo do setor e os códigos de checagem de erro.

Tabela 4.2: Comandos ATA selecionados em ordem alfabética

Comando	Código	Descrição
Read Sector	20h	Usado para ler o conteúdo de um ou mais setores
Read Sector Ext	24h	Usado para ler o conteúdo de um ou mais setores
Read Verify Sectors	40h	Usado para verificar o conteúdo de um ou mais setores
Read Verify Sectors Ext	44h	Usado para verificar o conteúdo de um ou mais setores
SMART	B0h	Este comando implementa diferentes funcionalidades relacionadas com o sistema SMART, que foram implementadas em diferentes classes no *framework* proposto. São eles: ▶ Diasable Operations (B0h/D9h) ▶ Enable Operations (B0h/D8h) ▶ Execute off-line immediate (B0h/D4h), este subcomando executa os auto-testes SMART, que são determinados através dos parâmetros passados na estrutura do comando. ▶ Read Data (B0h/D0h) ▶ Read Log (B0h/D5h) ▶ Return Status (B0h/DAh)
Write Sector	30h	Usado para escrever dados em um ou mais setores
Write Uncorrectable	45h	Usado para inserir erros

in off-line mode.

- *Linear Seek* executa leituras com o comando READ VERIFY de modo que as leituras sejam feitas com o mesmo espaçamento, como descrito na Figura 4.4.a. O espaçamento utilizado é calculado em função do tamanho do disco e 5000 setores são analisados. Se algum setor falho for encontrado o teste será encerrado.
  - *Linear Seek 1* Inicia o teste da menor para a maior LBA.
  - *Linear Seek 2* Inicia o teste da maior para a menor LBA.
- *Funnel Seek* executa leituras com o comando READ VERIFY, de modo que as leituras sejam feitas em dois sentidos, o algoritmo alternará leituras da menor LBA para a maior e da maior para a menor LBA , como descrito na Figura 4.4.b. Se algum setor falho for encontrado o teste será encerrado.
  - *Funnel Seek 1* Neste algoritmo são utilizados dois passos, um passo fixo de 350.000 LBAs e um passo aleatório limitado a 10.000 LBAs. Uma variável é utilizada para monitorar a condição de parada. A cada execução um número de LBA é gerado somando a posição atual, iniciada em zero, o passo fixo e o passo aleatório. Esta LBA é analisada, primeiro sentido, e também a LBA correspondente ao valor total de LBAs menos o número gerado, segundo sentido. O teste termina quando a LBA gerada no primeiro sentido for maior ou igual ao valor total de LBAs.
  - *Funnel Seek 2* Realiza o mesmo procedimento do *Funnel Seek 1*, entretanto no início do teste as 100 primeiras LBAs são analisadas e no final do teste a primeira e a última LBAs são analisadas.
- *Random Seek* executa leituras com o comando READ VERIFY de modo que um percentual do disco seja analisado e as leituras sejam feitas de maneira aleatória, como descrito na Figura 4.5.a. Se algum setor falho for encontrado o teste será encerrado.
  - *Random Seek 1* executa leituras em 5000 setores.
  - *Random Seek 2* executa leituras em 7500 setores.
  - *Random Seek 3* executa leituras em 10000 setores.

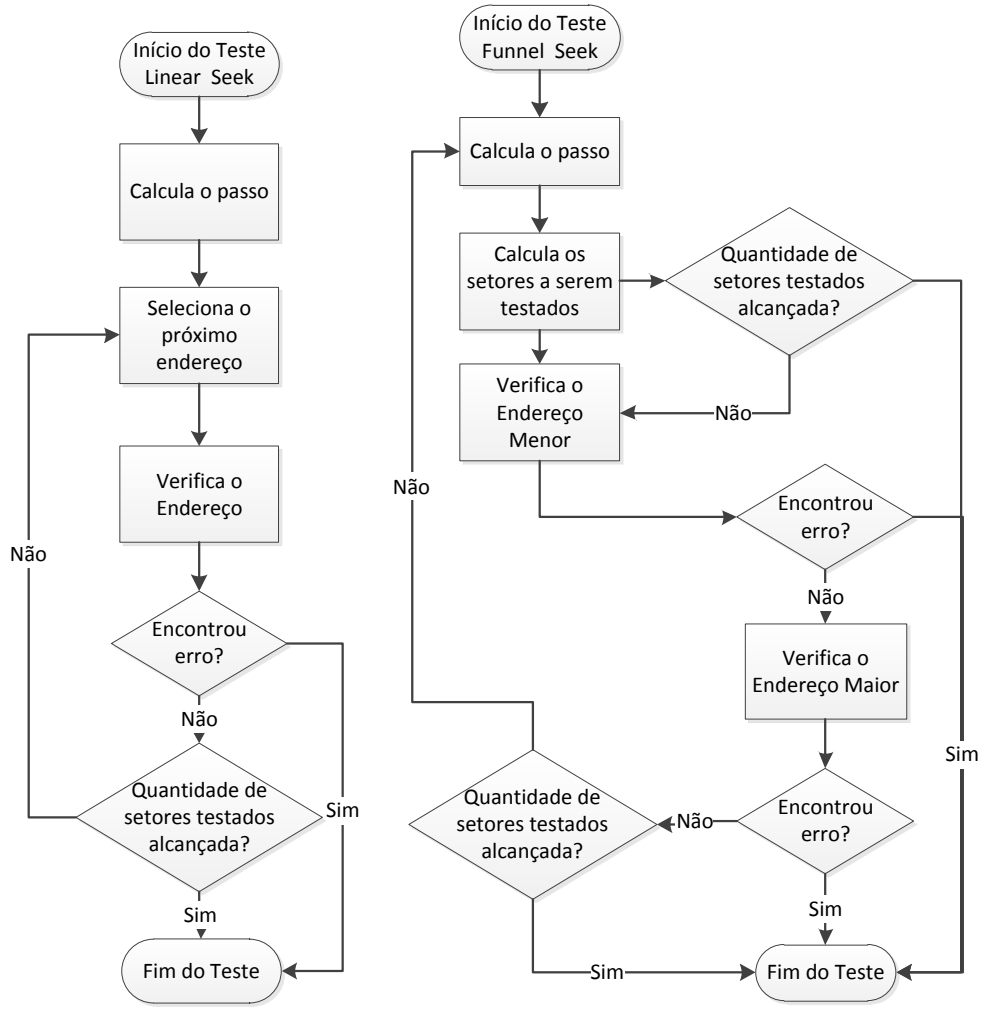
- *Surface Scan* executa leituras com o comando READ VERIFY. As leituras são feitas com o mesmo espaçamento para cada passo dado. Um grupo de setores ou trecho da superfície é analisado, como descrito na Figura 4.5.b. Se algum setor falho for encontrado o teste será encerrado.
  - *Surface Scan 1* Divide o Disco em 2.000 "lotes" de setores e para cada início de "lote" 30 setores são analisados. O algoritmo começa executando da menor para a maior LBA.
  - *Surface Scan 2* Mesmo procedimento do *Surface Scan 1*, mas segue o sentido oposto, começa executando da maior para a menor LBA.
- *Targeted Read Test* este teste realiza a leitura do *log* de falhas da controladora, identifica os setores listados e realiza novas leituras, nos setores e na sua vizinhança, para determinar se o mesmo continua com falhas. Cada LBA encontrado nos *logs* é analisada e também os 5 setores anteriores e posteriores a ele. Este teste é comentado em (**??**) e descrito na Figura 4.6. O teste é finalizado quando todos os setores listados ou quando mais de 5 setores falhos são encontrados.

As quantidades de setores testadas em cada teste foram escolhidas com base no tempo médio de execução, avaliado em testes pré-eliminares em HDDs (que são mais lentos que SSDs). Neste testes observou-se que o tempo médio para leitura de 5000 setores, em vários *desktops* e *laptops*, variava de 50 a 60 segundos, tempo próximo ao observado nos *softwares* do mercado para cada teste realizado.

4.2.4 *Bootable*

Utilizou-se um sistema Debian GNU/Linux para criar uma versão inicializável do Linux. Ele foi criado a partir de uma versão de 64 bits do Debian 6.0 Squeeze, kernel 2.6.32-5-amd64, usando o sistema de arquivos *squashfs*. Este sistema de arquivos é somente de leitura e implementa compressão, sendo o mais indicado para versões inicializáveis, *live cd* ou *pendrive* (**??**).

No total, o *bootable* ocupa cerca de 200 MB e pode ser utilizado em qualquer computador que suporte inicialização por cd ou *pendrive*. O processo de criação e atualização do *bootable* é descrito em (**??**). Uma "imagem" deste sistema de arquivos, com o Debian Squeeze instalado, foi modificada e nela o binário gerado na compilação



(a) Linear Seek (b) Funnel Seek

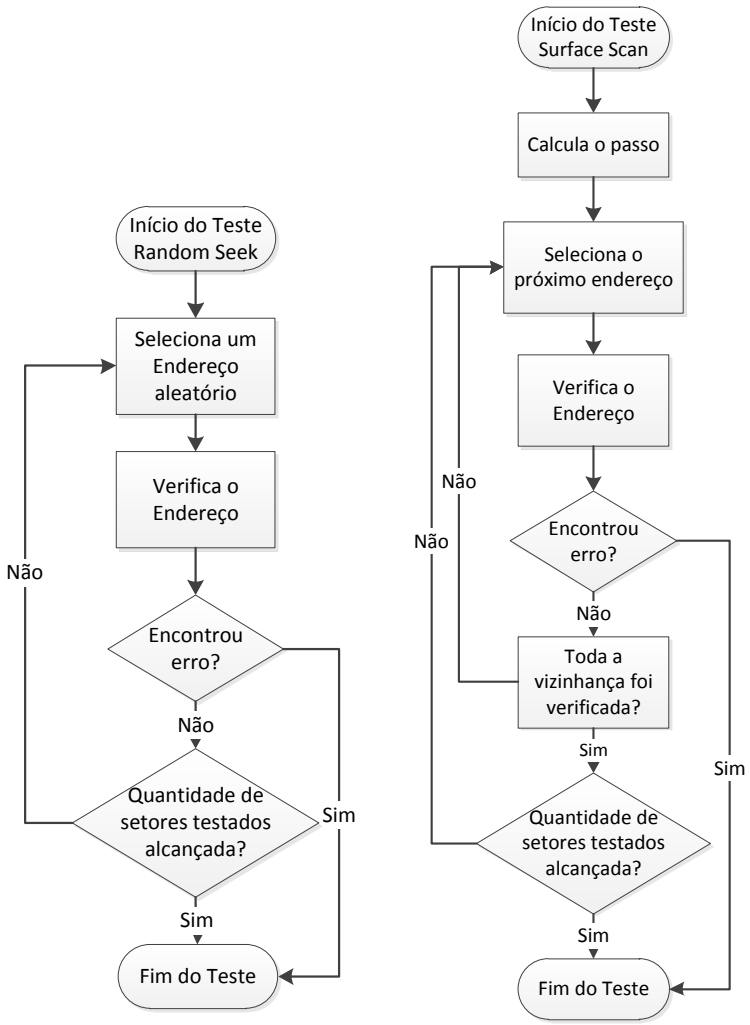
Figura 4.4: Fluxograma dos algoritmos implementados: *Linear e Funnel Seek*

da camada de aplicação foi inserido. Esta nova "imagem" é então copiada para *pendrives* ou mídias ópticas, como CDs ou DVDs, e está pronta para ser executada.

O binário gerado na camada de aplicação pode ser executado pelo *bootable*, através da reinicialização do sistema, e pelo terminal, como um programa comum em outros sistemas Linux em execução, como Ubuntu, CentOS e outros, sem requerer reinicialização. Nos dois casos os parâmetros são passados por linha de comando.

4.3 Resumo do Capítulo

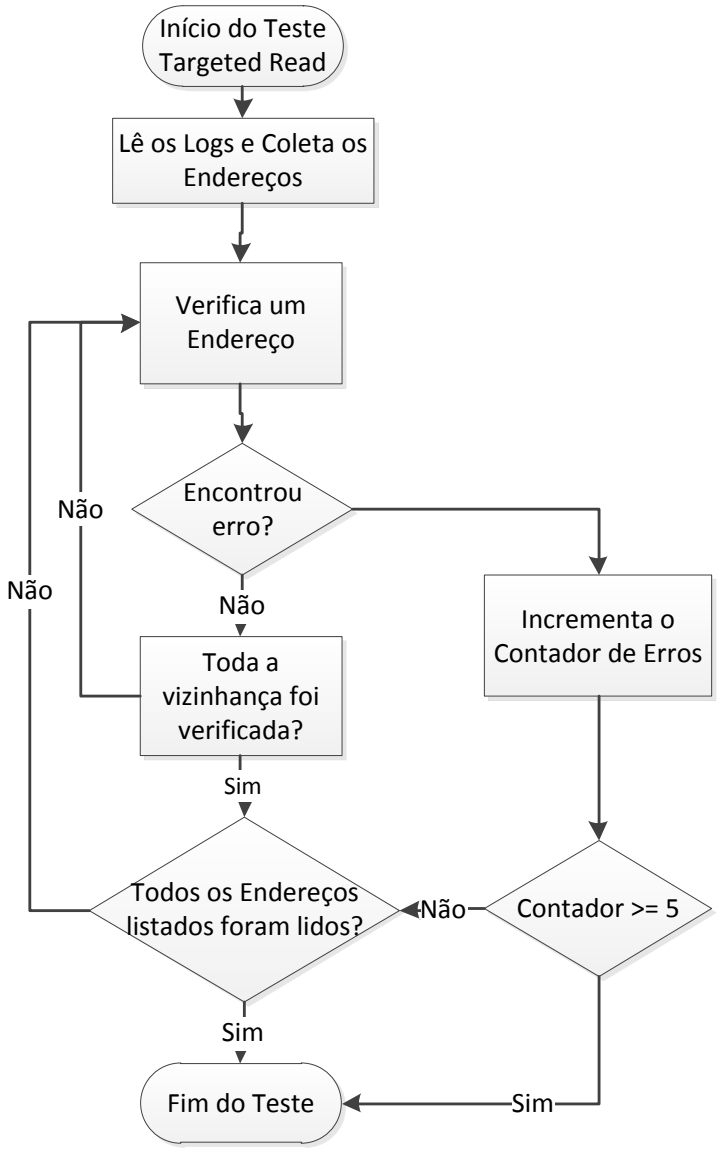
Este capítulo detalhou a implementação do *framework* e algoritmos propostos, bem como as razões pelos quais foram escolhidos. No próximo capítulo os resultados



(a) Random Seek (b) Surface Scan

Figura 4.5: Fluxograma dos algoritmos implementados: *Random Seek e Surface Scan*

do algoritmos implementados serão comparados com os testes de ferramentas de diagnóstico do mercado.



(a) Targeted Read

Figura 4.6: Fluxograma dos algoritmos implementados: *Targeted Read*

Capítulo 5

Resultados

Este capítulo aborda os resultados deste trabalho. Entre eles estão a camada de aplicação, os resultados obtidos a partir da metodologia descrita na Seção 4.1, assim como sua análise temporal.

5.1 Camada de Aplicação

A implementação da camada de aplicação é um programa que compreende, além dos algoritmos de teste, uma interface para que o usuário possa executá-los e para isso foram implementadas funções de listagem dos discos presentes no sistema e a filtragem para que o usuário possa escolher em que disco deseja realizar testes.

A Figura 5.1 mostra a interface inicial da implementação da camada de aplicação. Ela contém o título do programa, as listagens dos algoritmos e dispositivos disponíveis. Entre colchetes estão os parâmetros que devem ser passados para o programa durante a execução de cada teste. Essa figura demonstra a execução do programa sem a passagem de parâmetros, isso faz com que o mesmo realize a listagem das opções disponíveis.

O usuário pode obter informações sobre os parâmetros esperados executando a opção **-h** ou –**help**. A tela exibida durante a ajuda é mostrada na Figura 5.2.

A Figura 5.3 mostra um algoritmo sendo executado. Para executar um algoritmo, os seguintes parâmetros devem ser passados: **-d** ou **- -device** "disco" **-a** ou **- -algorithm** "algoritmo". No caso da figura os parâmetros passados foram: **-d /dev/sdb -a short**. O programa então passa a exibir uma barra de progresso

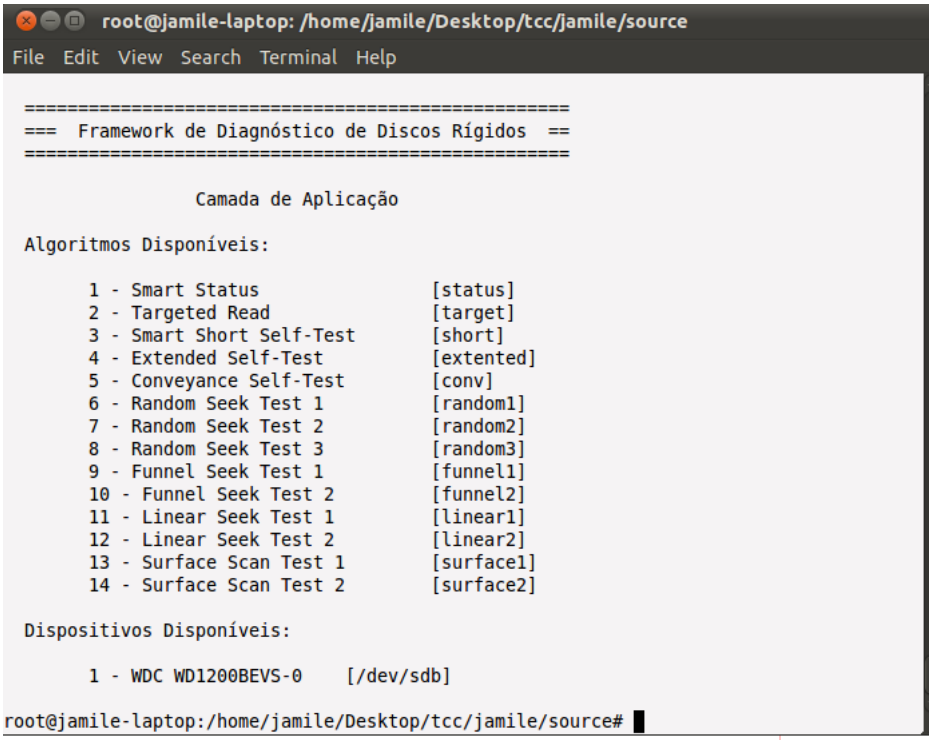


Figura 5.1: Tela inicial da Camada de Aplicação

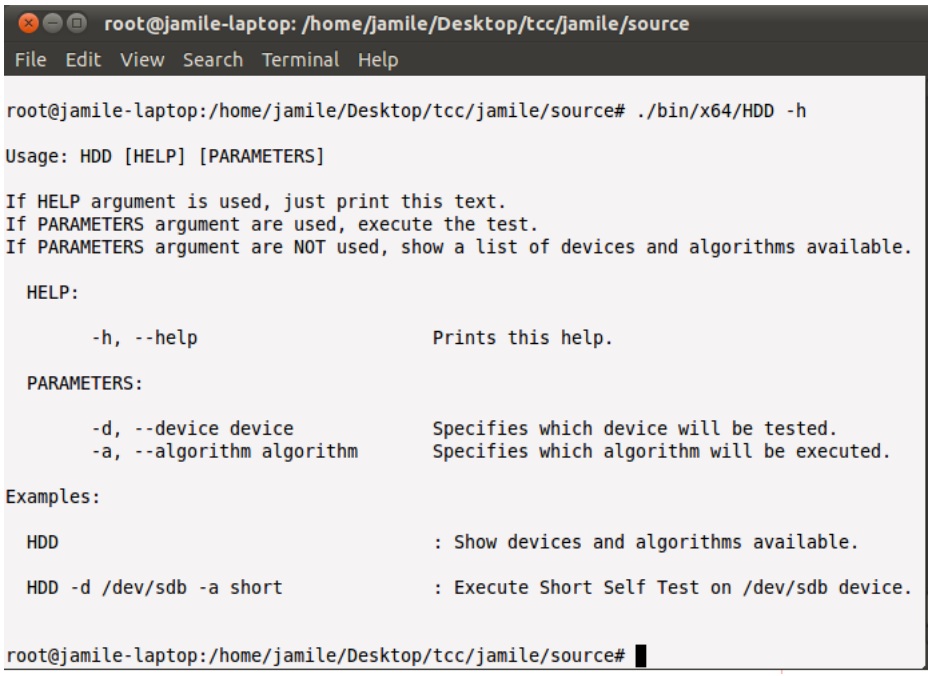


Figura 5.2: Tela de ajuda

com o percentual do teste concluído, qual algoritmo está sendo executado e em que dispositivo.

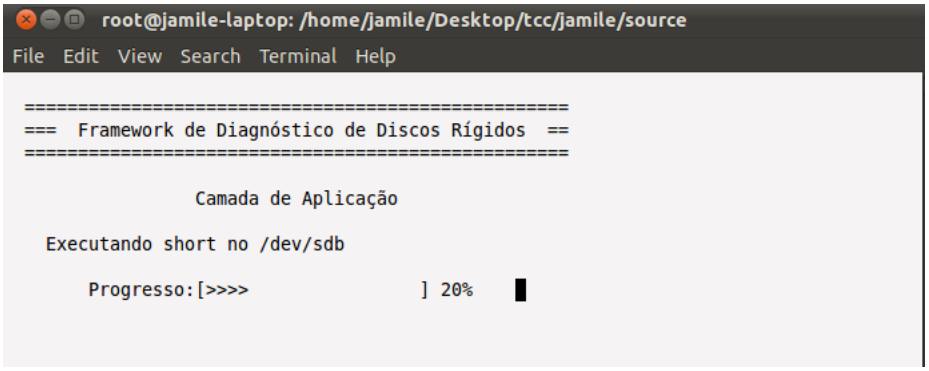


Figura 5.3: Tela de teste em execução

Na conclusão do teste, o tempo de execução é mostrado e o resultado do teste é apresentado, como mostrado na Figura 5.4. O teste mostrado levou 120 segundos e foi concluído com sucesso.

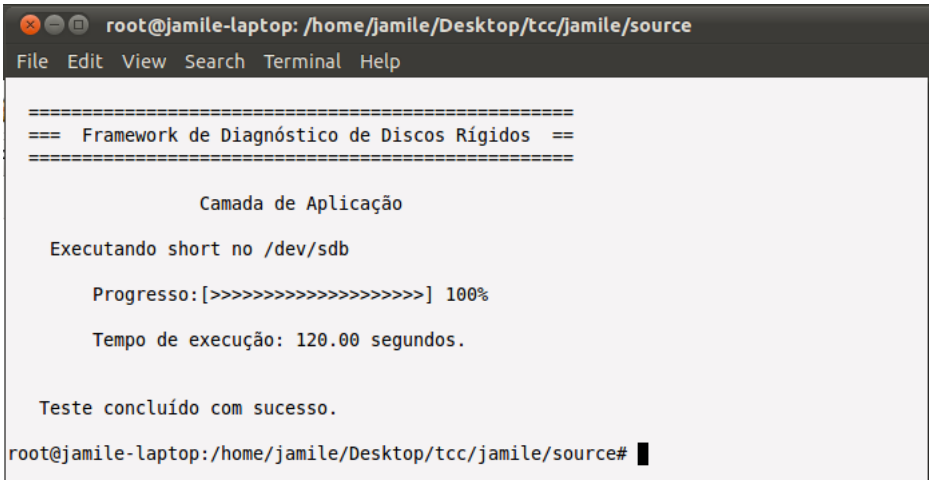


Figura 5.4: Tela de teste concluído

5.2 Testes

Nesta seção são apresentados os resultados obtidos com a metodologia descrita no capítulo anterior. Para a realização dos testes foram utilizados 12 discos rígidos, 9 do tipo HDD (H1, H2, H3, H4, H5, H6, H7, H8 e H9) e 3 do tipo SSD (S1, S2 e S3), com defeitos conhecidos. Cada teste descrito nas tabelas é o resultado obtido na maior parte de 3 execuções, ou seja, o valor apresentado se repetiu de duas a três vezes. A notação utilizada em todas as tabelas de resultados é descrita na Tabela 5.1. Na Tabela 5.2, os dispositivos testados são listados assim como modelos, *form factors*, capacidades de armazenamento e o resultado obtido testando o dispositivo

com outro *software* de diagnóstico.

Tabela 5.1: Notação utilizada nas tabelas de resultados

Abreviação / Símbolo	Definição
×	Teste falhou
✓	Teste passou
∅	Teste não pôde ser executado
Short	SMART *Short Self-Test*
Status	SMART *Return Status*
Conv	SMART *Conveyance Self-Test*
Linear1	*Linear Seek Test 1*
Linear2	*Linear Seek Test 2*
Random1	*Random Seek Test 1*
Random2	*Random Seek Test 2*
Random3	*Random Seek Test 3*
Funnel1	*Funnel Seek Test 1*
Funnel2	*Funnel Seek Test 2*
Surface1	*Surface Scan Test 1*
Surface2	*Surface Scan Test 2*
Target	*Targeted Read Test*

Uma sequência de execução dos algoritmos foi definida com base nos *softwares* do mercado e uma série de hipóteses foi levantada. O primeiro questionamento é relativo à execução do *Targeted Read Test*, teste que realiza leituras específicas em regiões onde algum tipo de problema foi detectado, a fim de determinar qual a melhor ordem de execução, no início ou no final dos testes.

O segundo questionamento é em relação à quantidade de setores checados no algoritmo *Random Seek*. Para responder a este questionamento, o desempenho de três taxas distintas é avaliado.

O terceiro questionamento é quanto à validade da distinção feita nos algoritmos de *Surface Scan* e *Linear Seek*. Foi avaliado se realizar os testes indo das menores para as maiores LBAs, ou das maiores para as menores LBAs, apresentam variações significativas de desempenho.

Por último, uma hipótese é levantada sobre o funcionamento do teste *Funnel*

Tabela 5.2: Lista dos dispositivos testados

Dispositivo	Modelo	Capacidade	*Form Factor*	PC-Doctor
H1	Seagate ST3500418AS	500 GB	3.5”	✓
H2	Western Digital WD2500AAKX083CA0	250GB	3.5”	×
H3	Seagate ST31000528AS	1 TB	3.5”	×
H4	Seagate ST3500418AS	500 GB	2.5”	✓
H5	Hitachi HTS725050A9A364	500 GB	2.5”	×
H6	Hitachi HTS545032B9A300	320 GB	2.5”	×
H7	Seagate ST9250315AS	250 GB	2.5”	×
H8	Toshiba MK5061GSY	500GB	2.5”	×
H9	Hitachi HTS723232A7A364	320 GB	2.5”	✓
S1	Kingston SNV425S264GB	64 GB	2.5”	✓
S2	Toshiba THN5NC128GCSJ	160 GB	2.5”	×
S3	Kingston SV100S264G	64 GB	2.5”	✓

Seek. Saber se, além da alternância no sentido de “crescimento” da LBA analisada, a leitura de setores mais específicos, como os setores iniciais do disco, que contém informações sobre a tabela de partição[1], pode influenciar no desempenho do algoritmo.

A ordem de execução dos testes definida foi: *Target Read Test*, SMART *Status Test*, SMART *Short Self-Test*, SMART *Conveyance Self-Test*, *Random Seek* 1, 2 e 3, *Funnel Seek* 1 e 2, *Linear Seek* 1 e 2, *Surface Scan* 1 e 2, e *Target Read Test* e os resultados são apresentados na Tabela 5.3.

Os melhores resultados obtidos individualmente foram *SMART Short Self-Test* e *Targeted Read Test*, que detectaram 6 dos 7 dispositivos com falhas. Depois destes estão o *Funnel Seek 2* e o *SMART Conveyance Self-Test*, que detectaram sozinhos 3 dos 7 dispositivos falhos. Entretanto, vale salientar que o *SMART Conveyance Self-Test* não pode ser executado na maioria dos dispositivos, pois estes não têm suporte a este tipo de teste. Em seguida está o *Surface Scan 2*, que detectou 2 dos 7 dispositivos falhos e os testes *Random Seek 2*, *Random Seek 3*, *Funnel Seek*

[1] *Master Boot Record* (MBR), setor que contém a tabela de partições do disco e informações sobre a inicialização do sistema operacional, localizado no setor 0.

Tabela 5.3: Resultados dos Testes.

Algoritmos	H1	H2	H3	H4	H5	H6	H7	H8	H9	S1	S2	S3
Status	✓	✓	×	✓	✓	✓	✓	✓	✓	✓	✓	✓
Short	✓	×	×	✓	×	✓	×	×	✓	✓	×	✓
Conv	✓	×	×	✓	∅	∅	×	∅	∅	∅	∅	∅
Random1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Random2	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓	✓
Random3	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓	✓
Funnel1	✓	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓
Funnel2	✓	×	✓	✓	×	✓	×	✓	✓	✓	✓	✓
Linear1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Linear2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Surface1	✓	×	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Surface2	✓	×	✓	✓	✓	✓	✓	×	✓	✓	✓	✓
Target	✓	×	✓	✓	×	×	×	×	✓	✓	×	✓

1 e *Surface Scan 1* que detectaram apenas 1 dos 7 dispositivos falhos. Por último, estão *Random Seek 1*, *Linear Seek 1* e *Linear Seek 2*, que não detectaram nenhum dispositivo falho. O alerta de falha dado pelo *SMART Return Status* foi dado apenas para 1 dispositivo e em nenhum dos testes foi observado resultados "falsos positivos".

Há importantes pontos a serem analisados. Das três rodadas de testes executadas, a primeira apresentou resultados diferentes entre a execução do *Targeted Read Test* no início e no final dos testes. A execução inicial detectou apenas 3 dos 7 dispositivos com falhas. Depois da execução dos demais testes o *Targeted Read Test* passou a detectar 6 dos 7 dispositivos com falha. Este resultado, da execução final, se repetiu nas duas execuções de cada uma das duas rodadas seguintes. Logo, para o usuário é interessante que este teste seja o último a ser executado.

Este comportamento do *Targeted Read Test* pode ser explicado pela leitura dos *logs* da controladora. Quando outros testes são executados, setores com falha podem ser detectados ou, mesmo que o teste não seja reprovado, qualquer comportamento estranho que tenha ocorrido será registrado no *log* e posteriormente verificado com o *Targeted Read Test*.

O *Random Seek Test* apresentou baixo desempenho na detecção de dispositivos

com falha, apenas os testes 2 e 3 detectaram 1 dos 7 dispositivos falhos. Entretanto, o desempenho dos algoritmos deve ser analisado conjuntamente e como se trata de um algoritmo fundamentado em aleatoriedade, a realização destes testes com outro conjunto de dispositivos pode apresentar resultados variáveis.

Os algoritmos de *Linear Seek* e *Surface Scan* não se mostraram eficientes independentemente da ordem de "crescimento" das LBAs analisadas, se da menor para a maior ou o contrário.

Um desempenho interessante foi o do *Funnel Seek Test 2*, que se mostrou significativamente melhor que o *Funnel Seek 1*, lembrando que a diferença entre eles é a checagem dos 100 primeiros setores do disco feita pelo *Funnel Seek Test 2*.

De maneira geral, o conjunto dos 13 testes implementados foi capaz de detectar os 7 dispositivos defeituosos, alcançando assim o resultado obtido utilizando o PC-Doctor.

5.3 Tempo de Execução

Além da cobertura de falhas, o tempo de execução dos testes deve ser analisado. Na Tabela 5.4, são apresentados os tempos de execução da última rodada dos testes.

Na tabela há duas grandes "discrepâncias" de tempo: a primeira ocorre entre o tempo levado para executar os testes em dispositivos SSD e a segunda ocorre quando uma falha é encontrada. Neste caso, em uma leitura normal leva-se um tempo maior para o resultado da leitura, pois várias tentativas são feitas. Entretanto, quando um setor falho é encontrado, a condição de parada do algoritmo é satisfeita e ele é encerrado. Como no teste de *Surface Scan 1* do H2, que durou 7 segundos, enquanto que o mesmo teste para o H1, que tem o dobro da capacidade, levou 535 segundos, por exemplo.

Os testes mais rápidos foram os de *SMART Return Status* e *Targeted Read*. Em vários dispositivos a execução levou menos de 1 segundo. Entre os testes de *Random Seek* os testes variaram de 82 a 274 segundos para HDDs e entre 1 e 6 para SSDs. Os testes mais demorados foram os de *Surface Scan*, chegando a durar 972 segundos, como no caso do H9. Na Figura 5.5, o tempo médio em segundos de cada teste realizado em dispositivos HDD (mais demorados) é apresentado.

Uma seleção de algoritmos foi feita com base na análise dos resultados dos testes

Tabela 5.4: Tempos de Execução dos Testes, em segundos.

Algoritmos	H1	H2	H3	H4	H5	H6	H7	H8	H9	S1	S2	S3
Status	0	1	0	0	0	0	14	0	0	0	1	0
Short	81	10	10	60	50	120	24	41	120	50	11	50
Conv	140	10	10	121	-	-	24	-	-	-	-	-
Random1	106	82	107	106	126	131	114	83	137	3	1	1
Random2	159	123	162	160	192	196	163	125	205	4	3	2
Random3	214	163	250	213	192	262	213	166	274	6	4	4
Funnel1	129	52	524	129	150	101	18	103	104	0	0	0
Funnel2	131	52	400	130	150	102	19	104	106	0	1	1
Linear1	49	31	52	47	71	84	67	43	89	3	2	2
Linear2	48	32	52	48	72	84	66	44	89	3	2	2
Surface1	535	7	541	533	848	872	677	501	972	36	19	19
Surface2	536	71	543	533	852	873	678	204	968	36	19	19
Target	1	27	0	0	32	24	10	14	1	0	0	0

e dos tempos de execução. Os cinco algoritmos selecionados foram: *SMART Return Status*, *SMART Short Self-Test*, *Random Seek Test 2*, *Funnel Seek Test 2* e *Target Read Test*. Considerando apenas esta combinação de algoritmos foi possível detectar todos os dispositivos com falha e a soma dos tempos de execução foi inferior aos 10 minutos colocados como objetivo para a execução de um teste rápido e eficiente no diagnóstico de discos rígidos. Na Tabela 5.5, os resultados obtidos com esta seleção de algoritmos e o PC-Doctor foram comparados. Todos os dispositivos defeituosos detectados pelo PC-Doctor também foram detectados pela seleção de algoritmos.

Tabela 5.5: Comparação dos Resultados obtidos com a seleção de algoritmos e o PC-Doctor.

Algoritmos	H1	H2	H3	H4	H5	H6	H7	H8	H9	S1	S2	S3
Framework	✓	×	×	✓	×	×	×	×	✓	✓	×	✓
PC-Doctor	✓	×	×	✓	×	×	×	×	✓	✓	×	✓

O *SMART Return Status* foi escolhido por ser um importante sinalizador da "saúde" do dispositivo. O *SMART Short Self-Test* e o *Target Read Test* foram escolhidos por terem apresentado a melhor capacidade de detecção de falhas entre

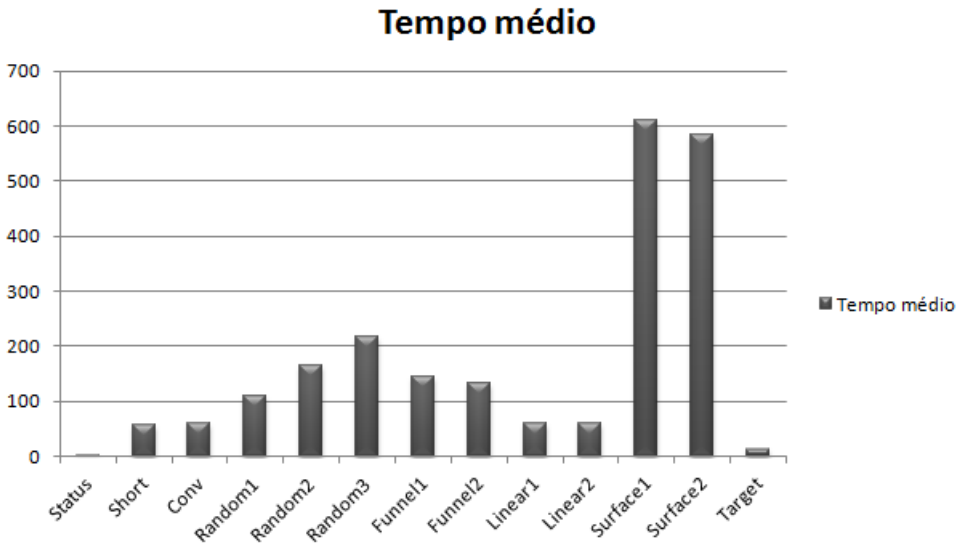


Figura 5.5: Tempo médio dos testes para dispositivos HDD

os algoritmos avaliados. Por fim, o *Random Seek Test 2* e o *Funnel Seek Test 2* foram escolhidos por terem apresentado uma capacidade mediana de detecção e servirem de complemento ao *Target Read Test*.

5.4 Resumo do Capítulo

Neste capítulo foram apresentados os resultados obtidos no desenvolvimento do *Framework* proposto: a criação de um programa capaz de executar testes de diagnóstico e que possibilitou a análise de algoritmos de teste e a seleção dos algoritmos mais eficientes com base na análise realizada.

No próximo capítulo são apresentadas as conclusões e propostas de continuação deste trabalho.

Capítulo 6

Conclusão

Neste trabalho foi desenvolvido um *framework* para dar suporte ao desenvolvimento de testes de diagnóstico de discos rígidos usados em computadores pessoais. Utilizou-se o sistema Linux pois, mesmo sendo empregado por uma pequena parcela do mercado, possibilita a realização de testes em computadores com qualquer sistema operacional instalado, bastando apenas a reinicialização.

O programa desenvolvido com a camada de aplicação, criada utilizando o *framework* apresentado neste trabalho, mostrou-se bastante eficiente pois apresentou desempenho similar aos das ferramentas de diagnóstico do mercado. Todos os dispositivos com falhas foram detectados, utilizando apenas uma combinação de 5 testes, com tempo de execução inferior a 10 minutos.

Entretanto, a maior contribuição deste trabalho está não apenas na análise realizada da eficiência dos algoritmos de testes, mas em fornecer os recursos necessários para a implementação de novos algoritmos de maneira simples, como os principais comandos necessários, uma camada de aplicação que encapsula estes comandos e um método de inserção de falhas, útil na validação de novos comandos e algoritmos de teste.

Durante o desenvolvimento deste trabalho, algumas dificuldades foram enfrentadas, principalmente devido à documentação dos padrões ATA e SCSI, que se mostrou superficial e mal organizada. Há livros que se dedicam a fazer uma análise mais detalhada destes padrões, entretanto os mesmos são antigos e estão um pouco defasados. Para contornar este empecilho, vários testes e experimentos foram realizados para tentar deduzir, através de investigação, o comportamento que não

era descrito pela documentação.

6.1 Perspectivas Futuras

Este trabalho pode ser continuado de diversas maneiras. Pode-se trabalhar na implementação de mais comandos SCSI, visando dar cobertura também a discos SAS e a servidores. A interface gráfica deve ser aprimorada, para que se torne mais robusta e mais agradável de ser utilizada. Por fim, a principal vertente de continuação deste trabalho é o desenvolvimento de novos algoritmos e o aprimoramento dos testes implementados aqui, com a realização de mais rodadas de teste e a experimentação de novos parâmetros em um conjunto maior de discos rígidos.