Trabalho Prático II – ENTREGA: 02 de DEZEMBRO de 2013

Implementação de um Sistema de Arquivos

1 Descrição Geral

O objetivo deste trabalho é a aplicação dos conceitos de sistemas operacionais na implementação de um Sistema de Arquivos capaz de gerenciar a ocupação de um disco físico, usando alocação indexada.

Esse Sistema de Arquivos será chamado, daqui para diante, de T2FS (*Task 2 – File System*) e deverá ser implementado, OBRIGATORIAMENTE, na linguagem "C" sem o uso de outras bibliotecas, com exceção da *libc* e da *libapidisk* (subsistema de E/S, que será fornecida pelo professor).

Além disso, a implementação deverá executar em ambiente UNIX.

O sistema de arquivos T2FS deve ser fornecido na forma de um arquivo de biblioteca chamada *libt2fs.a*. Essa biblioteca deverá fornecer uma interface através da qual programas de usuário e utilitários possam interagir com o sistema de arquivos. Comandos típicos aplicados sobre arquivos são o *create*, *open*, *read*, *write*, *close*, etc.

Na figura 1 está representado esse sistema. Nota-se a existência de três camadas de software: a camada inferior representa o acesso ao disco e é implementada pela *apidisk*. A camada superior representa os programas de usuário e utilitários de sistema. Essa camada é composta pelos programas desenvolvidos pelos usuários do T2FS, tais como os programas de teste escritos pelo professor e por vocês mesmos, e por programas utilitários (comando "*dir*", por exemplo). Finalmente, a camada intermediária representa o Sistema de Arquivos T2FS, e a implementação dessa camada é o objeto deste trabalho.

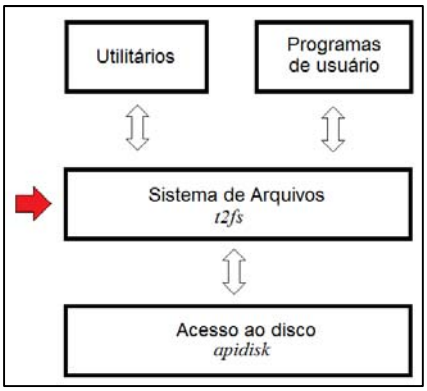

Figura 1 – Ambiente de uso do T2FS

A camada *apidisk* será fornecida pelo professor da disciplina. Essa camada é composta por um arquivo que simulará um disco lógico formatado em T2FS e por duas funções básicas para a leitura e para a escrita de blocos no T2FS. O arquivo que simula o disco lógico é análogo aqueles usados pelas máquinas virtuais, como por exemplo a VirtualBox.

2 Descrição do Sistema de Arquivos T2FS

O espaço disponível no disco está dividido em quatro áreas contíguas: área de superbloco, área de *bitmap* (para gerência dos blocos livres), área para o diretório raiz (*root*) e área para os blocos de índice e blocos de dados dos arquivos. A gerência do espaço em disco é feito por alocação indexada e o diretório segue uma estrutura linear de um único nível. Essas áreas estão representadas na figura 2 e estão detalhadas a seguir.

Área de superbloco: é a área de controle do sistema de arquivos. Essa área inicia, sempre, no bloco zero. Os blocos dessa área contêm os seguintes elementos, na ordem em que aparecem abaixo:

Posição relativa	Tamanho (bytes)	Nome	Descrição
0	4	*id*	Identificação do sistema de arquivo. É formado pelas letras "T2FS", ou seja, pelos valores ASCII 0x54 0x32 0x46 0x53
4	1	*version*	Versão atual desse sistema de arquivos. Deverá ser 0x01
5	1	*ctrlSize*	Número de blocos de controle

6	4	*diskSize*	Número de blocos totais do disco (formato *little endian*)
10	2	*blockSize*	Número de bytes de um bloco. Esse valor será, sempre, maior do que *fileEntrySize* (formato *little endian*)
12	2	*freeBlockSize*	Número de blocos da área usada para a gerência de espaço do disco (formato *little endian*)
14	2	*rootSize*	Número de blocos da área usada como diretório (formato *little endian*)
16	2	*fileEntrySize*	Número de bytes de um registro na área de diretório

Área de *bitmap* (gerência dos blocos livres): é a área onde estão os blocos usados para a gerência de espaço do disco, cujo tamanho está definido em *freeBlockSize*. Deve ser utilizado o mecanismo de *bitmap* para registrar a alocação dos blocos do disco, onde cada bit indica a alocação de um bloco do disco. O primeiro bit dessa área (correspondente ao bit 7 do primeiro byte) indica a alocação do bloco 0; o segundo bit (correspondente ao bit 6 do primeiro byte) indica a alocação do bloco 1; e assim por diante, até o último bloco. Se o bit for "0", então o bloco correspondente está livre; se o bit for "1", então o bloco correspondente está ocupado.

Área para o diretório raiz: área do disco posicionada logo após os blocos usados para gerência de espaço do disco. Possui tamanho (em blocos) definido por "*rootSize*". O diretório será organizado em uma estrutura linear de um único nível. Dessa forma, não existirão subdiretórios. Cada entrada (registro) que identifica um arquivo terá 64 bytes, distribuídos entre o nome do mesmo e ponteiros.

Área para os blocos de índice e blocos de dados: nessa área, que se estende até o final do disco, estão alocados os blocos de índice, usados para encontrar os blocos de dados dos arquivos, assim como os próprios blocos de dados.

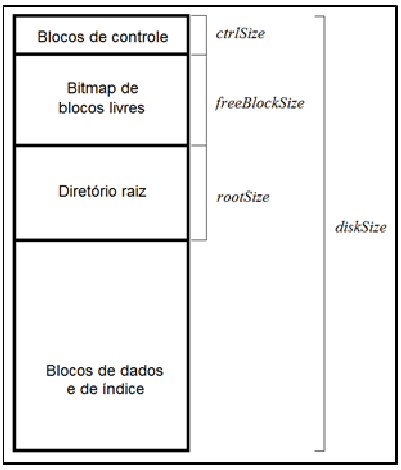

Figura 2 – Organização do disco lógico, segundo o T2FS

2.1 Estrutura interna do Diretório Raiz

Cada arquivo existente em um disco formatado em T2FS possui uma entrada (registro) no diretório raiz. Dessa forma, o diretório T2FS está estruturado com registros de tamanho fixo (64 bytes). Os arquivos são identificados através de seus registros na área do Diretório Raiz. Cada registro é formado pelo nome do arquivo, pelo seu tamanho em blocos, pelo seu tamanho em bytes e por um conjunto de 4 (quatro) ponteiros.

Nome do arquivo: ocupa os primeiros 40 bytes do registro. O nome dos arquivos em T2FS é formado por qualquer combinação dos caracteres ASCII 0x21 ("!") até 0x7A ("z"), sendo "*case sensitive*". Caso o nome do arquivo possua menos do que 40 caracteres, os bytes não ocupados deverão ser preenchidos com 0x00. Além disso, o bit 7 do primeiro caractere do nome do arquivo indica se o registro é válido: se for "1" (um), o registro é válido; se for "0" (zero), o registro não é válido e pode ser reutilizado para novos arquivos. Por registro válido, entende-se aqueles que estão associados a um arquivo existente no disco.

Tamanho em blocos: ocupa 4 (quatro) bytes e representa o tamanho do arquivo em blocos de dados. Notar que não está incluso nesse total os blocos utilizados para armazenar os índices.

Tamanho em bytes: ocupa 4 (quatro) bytes e representa o tamanho do arquivo em bytes. Notar que o tamanho de um arquivo não é um múltiplo do tamanho dos blocos de dados. Portanto, o último bloco de dados pode não estar totalmente utilizado. Assim, a detecção do término do arquivo dependerá da observação desse campo do registro.

Ponteiros: são 4 (quatro) ponteiros de 32 bits, armazenados em formato *little endian*, e ocupando os 16 bytes restantes do registro Esses ponteiros implementam a alocação indexada com três níveis: direto, indireção simples e indireção dupla, como segue:

- Os dois primeiros ponteiros apontam para blocos de dados do arquivo (ponteiro direto).
- O terceiro ponteiro aponta para um bloco do disco que contém índices (bloco de índices) de blocos de dados (ponteiro de indireção simples).
- Finalmente, o quarto ponteiro aponta para um bloco de índices que contém ponteiros para outros blocos de índice. Estes, por sua vez, contêm ponteiros para blocos de dados (ponteiro de indireção dupla).

Notar que a ocupação dos ponteiros é feita em ordem, iniciando pelos ponteiros de dados e terminando pelo quarto ponteiro. Caso os ponteiros não sejam usados, estes devem receber o valor 0 (zero).

Um registro (estrutura *t2fs_record*) tem a seguinte forma geral:

Posição relativa	Tamanho (bytes)	Nome	Descrição
0	40	*name*	Nome do arquivo que identifica o registro Bit de validade do registro (bit 7 do primeiro caractere do nome)
40	4	*blocksFileSize*	Tamanho do arquivo expresso em número de blocos (*little endian*).
44	4	*bytesFileSize*	Tamanho do arquivo expresso em número de bytes (*little endian*).
48	8	*dataPtr[2]*	Dois ponteiros diretos (*little endian*).
56	4	*singleIndPtr*	Ponteiro de indireção simples (*little endian*).
60	4	*doubleIndPtr*	Ponteiro de indireção dupla (*little endian*).

2.2 Formato de um bloco de índice

Os blocos de índice são blocos do disco onde estão armazenados ponteiros para blocos de dados ou outros blocos de índice, de um determinado arquivo. Cada ponteiro ocupa 4 (quatro) bytes e está armazenado em formato *little endian*.

A ocupação dos ponteiros deve ser feita em ordem, iniciando na posição 0 (zero) do bloco. Os ponteiros não usados devem receber o valor 0 (zero), sendo essa a forma de identificar o término da lista de ponteiros.

Dessa forma, levando-se em consideração a estrutura de ponteiros e o disco, o maior tamanho de um arquivo será dado através da seguinte expressão:

$$Tamanho = blockSize.\left(\left(\frac{blockSize}{4}\right)^2 + \left(\frac{blockSize}{4}\right) + 2\right)$$

3 Especificação da *t2fs* (libt2fs.a)

Sua tarefa é implementar a biblioteca *libt2fs.a*, que possibilitará a leitura e escrita de arquivos nesse sistema de arquivos. As funções a serem implementadas estão resumidas na tabela abaixo e são detalhadas logo a seguir.

Nome	Descrição
`char *t2fs_identify (void)`	Retorna a identificação dos implementadores do T2FS.
`t2fs_file t2fs_create (char *nome)`	Função usada para criar um novo arquivo no disco.
`int t2fs_delete (char *nome)`	Função usada para remover (apagar) um arquivo do disco.
`t2fs_file t2fs_open (char *nome)`	Função que abre um arquivo existente no disco.
`int t2fs_close (t2fs_file handle)`	Função usada para fechar um arquivo.
`int t2fs_read (t2fs_file handle, char *buffer, int size)`	Função usada para realizar a leitura em um arquivo.
`int t2fs_write (t2fs_file handle, char *buffer, int size)`	Função usada para realizar a escrita em um arquivo.
`int t2fs_seek (t2fs_file handle, unsigned int offset)`	Altera o contador de posição (*current pointer*) do arquivo.
`int t2fs_first (t2fs_find *findStruct)`	Inicializa uma sessão de pesquisa no diretório.
`int t2fs_next (t2fs_find *findStruct, t2fs_record *dirFile)`	Realiza a leitura do próximo registro do diretório.

Nessa tabela são usados alguns tipos de dados e protótipos de função que estão definidos no arquivo *t2fs.h*. Os protótipos de função e os tipos de dados existentes nesse arquivo de inclusão (*header files* ou arquivo ".h") deve seguir, rigorosamente, o especificado neste documento. Esse arquivo de inclusão será fornecido pelo professor. A **exceção** é o tipo *t2fs_find*, que deve ser definido pelo aluno e acrescentados ao arquivo *t2fs.h*.

A implementação do sistema de arquivos T2FS deve ser feita de tal forma que seja possível ter-se até 20 (vinte) arquivos abertos simultaneamente.

As funções *t2fs_create*, *t2fs_delete*, *t2fs_open*, *t2fs_close*, *t2fs_read*, *t2fs_write* e *t2fs_seek* são aquelas tipicamente empregadas por usuários ao desenvolverem aplicativos que manipulam arquivos. As funções *t2fs_first* e *t2fs_next* são funções auxiliares usadas para manipulação de diretórios.

```
char *t2fs_identify (void)
```

Função usada para identificar os implementadores do T2FS. Essa função deve retornar um ponteiro para um *string* formado apenas por caracteres ASCII (Valores entre 0x20 e 0x7A) e terminado por '\0' com a identificação dos implementadores: nome e número do cartão.

```
t2fs_file t2fs_create (char *nome)
```

Função que cria um novo arquivo. O nome desse novo arquivo é aquele informado pelo parâmetro "**nome**". O contador de posição do arquivo (*current pointer*) deve ser colocado na posição zero. Além disso, caso o arquivo já exista, ele será substituído por um arquivo vazio.

A função deve retornar o identificador (*handle*) do arquivo, que será usado em chamadas posteriores do sistema de arquivo para fins de manipulação do arquivo criado.

Caso ocorra erro deve retornar um valor negativo.

```
int t2fs_delete (char *nome)
```

Função usada para apagar um arquivo do disco. O nome do arquivo a ser apagado é aquele informado pelo parâmetro "**nome**".

Se a operação foi realizada com sucesso, a função retorna "0" (zero). Em caso de erro, será retornado um valor diferente de zero.

```
t2fs_file t2fs_open (char *nome)
```

Função que abre um arquivo existente no disco. O nome desse novo arquivo é aquele informado pelo parâmetro "**nome**". O contador de posição do arquivo (*current pointer*) deve ser colocado na posição zero.

A função deve retornar o identificador (*handle*) do arquivo, que será usado em chamadas posteriores do sistema de arquivo para fins de manipulação do arquivo.

Caso ocorra erro deve retornar um valor negativo.

```
int t2fs_close (t2fs_file handle)
```

Função que fecha o arquivo identificado pelo parâmetro "handle".

Se a operação foi realizada com sucesso, a função retorna "0" (zero). Em caso de erro, será retornado um valor diferente de zero.

```
int t2fs_read (t2fs_file handle, char *buffer, int size)
```

Função que realiza a leitura de "**size**" bytes do arquivo identificado por "**handle**". Os bytes lidos são colocados na área apontada por "**buffer**".

Após realizada a leitura, o contador de posição (*current pointer*) deve ser ajustado para o byte seguinte ao último lido.

Se a operação foi realizada com sucesso, a função retorna o número de bytes lidos. Notar que, se a função retornar zero, significa que o contador de posição atingiu o final do arquivo, sendo essa a forma de identificar esse evento (encerramento do arquivo).

Caso ocorra algum erro durante a execução da função, esta retornará um valor negativo.

```
int t2fs_write (t2fs_file handle, char *buffer, int size)
```

Função que realiza a escrita de "**size**" bytes no arquivo identificado por "**handle**". Os bytes a serem escritos estão na área apontada por "**buffer**".

Após realizada a escrita, o contador de posição (*current pointer*) deve ser ajustado para o byte seguinte ao último byte escrito.

Se a operação foi realizada com sucesso, a função retorna o número de bytes efetivamente escritos.

Caso ocorra algum erro durante a execução da função, esta retornará um valor negativo.

```
int t2fs_seek (t2fs_file handle, unsigned int offset)
```

Função usada para posicionar o contador de posições (*current pointer*) do arquivo identificado por "**handle**" na posição dada pelo parâmetro "**offset**". Esse valor corresponde ao deslocamento, em bytes, contados a partir do início do arquivo.

Se a operação foi realizada com sucesso, a função retorna "0" (zero). Caso ocorra algum erro, a função retorna um valor diferente de zero.

```
int t2fs_first (t2fs_find *findStruct)
```

Inicializa uma sessão de pesquisa no diretório e efetua a leitura do primeiro registro **válido** do diretório.

As informações de controle necessárias para que possam ser lidos os próximos registros do diretório através da função "*t2fs_next*" são colocadas na estrutura "**findStruct**".

Se a operação for realizada com sucesso, a função retorna "0" (zero). Caso ocorra algum erro, a função retorna um valor diferente de "0" (zero).

```
int t2fs_next (t2fs_find *findStruct, t2fs_record *dirFile)
```

Obtém o próximo registro **válido** de diretório.

As informações de controle necessárias para identificar qual o registro a ser retornado estão armazenadas na estrutura "**findStruct**", obtida através de uma chamada anterior à função "*t2fs_first*". As informações do registro do diretório lido são colocadas na estrutura "**dirFile**".

Se a operação for realizada com sucesso, a função retorna "0" (zero). Se não foi possível retornar um novo registro porque o diretório chegou ao final, a função retorna "1" (um). Caso ocorra algum erro, a função retorna um valor diferente de "0" (zero) e de "1" (um).

4 Interface da *apidisk* (libapidisk.a)

Um disco físico pode ser visto como uma estrutura tridimensional composta por cilindros (e trilhas) que, por sua vez, são divididos em setores com um número fixo de bytes. A tarefa do Subsistema de E/S é transformar essa estrutura tridimensional em uma sequência linear de blocos lógicos, numerados de 0 até L−1, onde L é o número total de blocos lógicos do disco físico.

Você receberá a biblioteca *apidisk* que realiza as operações desse Subsistema de E/S. Portanto, esse Subsistema de E/S (a biblioteca *apidisk*) permitirá a leitura e a escrita dos blocos lógicos do disco, que serão endereçados através de sua numeração sequencial.

Esse Subsistema de E/S realizam as operações de leitura e escrita de blocos lógicos do disco, através das funções descritas a seguir.

```
int read_block (unsigned int bloco, char *buffer)
```

Realiza a leitura do bloco "`bloco`" do disco e coloca os bytes lidos no espaço de memória indicado pelo ponteiro "`buffer`".

Retorna "0", se a leitura foi realizada corretamente e um valor diferente de zero, caso tenha ocorrido algum erro.

```
int write_block (unsigned int bloco, char *buffer)
```

Realiza a escrita do conteúdo da memória indicada pelo ponteiro "`buffer`" no bloco "`bloco`" do disco.

Retorna "0", se a escrita foi bem sucedida; retorna um valor diferente de zero, caso tenha ocorrido algum erro.

A biblioteca *libapidisk.a*, que implementa as funções *read_block*() e *write_block*(), e o arquivo de inclusão *apidisk.h*, com os protótipos dessas funções, serão fornecidos pelo professor. Além disso, será fornecido um arquivo de dados para emulação do disco onde estará o sistema de arquivos T2FS.

5 Geração da *libt2fs*

As funcionalidades do sistema de arquivos T2FS deverão ser disponibilizadas através de uma biblioteca de nome *libt2fs.a*. Para gerar essa biblioteca deverá ser utilizada a seguinte "receita":

1. Cada arquivo "<*file_name.c*>" deverá ser compilado com a seguinte linha de comando:

```
gcc -c <file_name.c> -Wall
```

2. Todos os arquivos compilados (representados por *file_1*, *file_2*, ...) e a biblioteca *libapidisk.a* devem ser ligados usando a seguinte linha de comando:

```
ar crs libt2fs.a <file_1>.o <file_2>.o ... libapidisk.a
```

Para ambas as etapas: compilação e geração da biblioteca, não deverão ocorrer mensagens de erro ou de "*warning*".

Para fins de teste, a biblioteca *libt2fs.a* deverá ser ligada com o programa chamador. Para fazer a ligação deve-se utilizar a seguinte linha de comando:

```
gcc -o <nome_exec> <nome_chamador.c> -L<lib_dir> -lt2fs -Wall
```

onde <*nome_exec*> é o nome do executável, <*nome_chamador.c*> é o nome do arquivo fonte onde é realizada a chamada das funções da biblioteca, <*lib_dir*> é o caminho do diretório onde está a bibliotecas "*libt2fs.a*".

6 Questionário

1. Sem alterar a quantidade de ponteiros de alocação indexada do Superbloco, quais outros fatores influenciam no maior tamanho de arquivo T2FS possível? Como esse fatores influenciam nessa tamanho?

2. Se fosse você reprojetar o T2FS para suportar um diretório organizado em árvore, que permite a criação de diretórios dentro de diretórios, que mudanças seriam necessárias na estrutura atual do T2FS? Responda explicando o impacto dessas alterações nas várias estruturas de controle do T2FS (Superbloco, área de *bitmap*, área de diretório, formato dos blocos de índice, etc) assim como nas funções da API (tabela seção 3).

3. Supondo que você desejasse melhorar o T2FS, permitindo a criação de vínculos estritos (*hardlinks*). Que alterações seriam necessárias no T2FS? Há necessidade da criação de novas funções? Se sim, quais? Se não, porque não.

4. As estruturas de controle do T2FS contêm informações que permitem verificar a consistência de alguns de seus elementos. Isso é possível graças a um certo nível de redundância de informação (por exemplo, no registro de arquivo, nas entradas do diretório, o número total de blocos usados por um arquivo e o tamanho do arquivo – em bytes – permitem uma verificação). Identifique quais outros elementos são redundantes e discuta como seria possível usar essa redundância para aumentar a confiabilidade do T2FS.

5. Como você implementou a atribuição dos identificadores de arquivos (*file handler*) pelas funções *t2fs_create* e *t2fs_open*? Discuta a questão da reutilização dos mesmos.

6. Como você implementou a gerência do contador de posição (*current pointer*) usado pela função *t2fs_seek*?

7. A escrita em um arquivo (realizada pela função *t2fs_write*) requer uma sequência de leituras e escritas de blocos de dados e de blocos de controle. Qual é a sequência usada por essa função? Se essa sequência for interrompida (por falta de energia, por exemplo) entre duas operações de escrita de bloco, qual será o efeito na consistência dos dados no disco? É possível projetar uma sequência de escritas no disco que minimize a eventual perda de dados?

8. Algumas estruturas gravadas no disco são mais facilmente manipuláveis se estiverem na memória principal (como se fosse uma *cachê*). Por outro lado, isso aumenta a possibilidade de perda de dados, pois as informações existentes nessa cachê e que não foram escritas no disco, podem ser perdidas, caso ocorra alguma interrupção de operação do sistema. Quais informações do disco você está mantendo (e gerenciando) na memória principal e porque você as escolheu? Qual a política que você usou para decidir quando escrevê-las no disco?

9. Todas as funções implementadas funcionam corretamente? Relate, para cada uma das funções desenvolvidas, como elas foram testadas?

10. Relate as suas maiores dificuldades no desenvolvimento deste trabalho e como elas foram contornadas.

7 Entregáveis: o que deve ser entregue?

Devem ser entregues:

- Todos os arquivos fonte (arquivos ".c" e ".h") que formam a biblioteca "*libt2fs*";
- Um arquivo *makefile* para criar a "*libt2fs.a*".
- O arquivo "*libt2fs.a*" e
- Um arquivo PDF com o relatório da implementação.

Os arquivos devem ser entregues em um *tar.gz* (SEM arquivos *rar* ou similares), segundo, **obrigatoriamente**, a seguinte estrutura de diretórios e arquivos:

\t2fs		
	makefile	ARQUIVO: arquivo makefile da "*libt2fs*" Deve possuir uma regra "*clean*", para limpar todos os arquivos gerados.
	relatorio.pdf	ARQUIVO: arquivo PDF com o relatório do trabalho
	include	DIRETÓRIO: local onde colocar todos os arquivo ".h". Nesse diretório deve estar o "*t2fs.h*". e o "*apidisk.h*"
	src	DIRETÓRIO: local onde colocar todos os arquivo ".c"
	bin	DIRETÓRIO: local onde será gerado o programa executável (junção da aplicação e a biblioteca "*libt2fs.a*")
	lib	DIRETÓRIO: local onde será gerada a biblioteca "*libt2fs.a*". (junção da "*t2fs*" com a "*libapidisk.a*"

No relatório da implementação devem estar presentes os seguintes elementos:

- Identificação dos componentes da dupla;
- Descrição do programa implementado (Não colocar o fonte no relatório) e das estruturas de dados utilizadas;
- Associação de cada elemento de implementação com o conceito correspondente visto nas aulas;
- Respostas ao questionário proposto (seção 6);

O trabalho deverá ser entregue até a **data prevista (02 de dezembro de 2013)**. Admite-se a entrega do trabalho com até uma semana de atraso. Nesse caso a nota final do trabalho será diminuída de 20,0 pontos (do total de 100,00). Não serão aceitos trabalhos entregues além dos prazos estabelecidos.

8 Avaliação

Para que um trabalho possa ser avaliado ele deverá cumprir com as seguintes condições:

- Entrega dentro dos prazos estabelecidos;
- Obediência à especificação (formato e nome das funções);
- Compilação e geração da biblioteca sem erros ou *warnings*;
- Fornecimento de todos os arquivos solicitados;
- O relatório deve estar completo.

Itens que serão avaliados e sua valoração:

- **1 ponto:** clareza e organização do código, programação modular, *makefiles*, arquivos de inclusão bem feitos (sem código C dentro de um include!!) e comentários adequados;
- **3 pontos:** resposta ao questionário e a correta associação entre a implementação e os conceitos vistos em aula;
- **6 pontos:** funcionamento do sistema de arquivos T2FS de acordo com a especificação. Para isso serão utilizados programas padronizados desenvolvidos pelo professor para essa verificação.

9 Data de entrega e avisos gerais – LEIA com MUITA ATENÇÃO!!!

1. Faz parte da avaliação a obediência RIGIDA aos padrões de entrega definidos na seção 7 (arquivos tar.gz, makefiles, estruturas de diretórios, etc)
2. O trabalho pode ser feito em DUPLAS (de dois! Duplas com mais de dois terão sua nota final dividida pelo número de participantes do grupo)
3. O trabalho deverá ser entregue até as 23:55:00 horas do dia 2 de dezembro via *moodle*. Entregar um arquivo *tar.gz* conforme descrito na seção 7.

4. Trabalhos entregues atrasados serão penalizados com desconto: entrega até 9 de dezembro de 2013 (23:55:00 horas) passa a valer no máximo 8 PONTOS. Esse será o atraso máximo permitido (uma semana). Expirado o atraso máximo, nenhum trabalho será mais aceito. ATENTEM para o fato que a prova de recuperação será feita no dia 11 de dezembro, portanto, para entregas atrasadas NÃO HÁ garantias quando a liberação da nota do trabalho ANTES da prova de recuperação. O prazo de 72 horas antes da avaliação é garantido apenas para os que entregarem o trabalho na data correta.

10 Observações

Recomenda-se a troca de ideias entre os alunos. Entretanto, a identificação de cópias de trabalhos acarretará na aplicação do Código Disciplina Discente e a tomada das medidas cabíveis para essa situação.

O professor da disciplina reserva-se o direito, caso necessário, de solicitar uma demonstração do programa, onde o aluno será arguido sobre o trabalho como um todo. Nesse caso, a nota final do trabalho levará em consideração o resultado da demonstração.