**UNIVERSIDADE FEDERAL DE SÃO CARLOS**

**Bacharelado em Ciência da Computação**

**Laboratório de Sistemas Operacionais**

**Professor Gustavo Maciel Dias Vieira**

***Campus* Sorocaba**

**Projeto Final**

**Sistema de Arquivos**

Bruna

Hitalo Siqueira

Rafael Paschoal Giordano - 408298

Thales Gonçalves Chagas - 408557

**Sorocaba**

**2014**

**Introdução**

O projeto final tem como objetivo estudar e entender a construção de software básico, construindo um sistema de arquivos elementar a partir de um dispositivo de disco genérico. O projeto é realizado através da implementação de funções de gerenciamento para esse sistema de arquivos, ajudando a compreender como funciona a abstração em arquivos dos sistemas operacionais.

**Descrição**

O sistema de arquivos foi implementado simulando o sistema de arquivos FAT, com estrutura definida de setores de 512 bytes e blocos de 8 setores com um total de 4096 bytes, com limite de 128 arquivos. Foram criadas as funções fornecidas pelo professor para manipular os arquivos, como criar, ler, remover, abrir, fechar e listar arquivos. AQUI PRECISA MUDAR SE NÃO IMPLEMENTAR TODAS

Abaixo segue a descrição do funcionamento dessas funções:

1. **int fs\_init()**

Esta função é responsável por inicializar o disco no sistema de arquivos. Esta é feita ao ler a FAT do disco para a memória, passando como parâmetro a posição de cada setor com o buffer auxiliar. Então é verificada a consistência da FAT, ou seja, se esta está formatada, para isso, de acordo com o modelo apresentado pelo professor, precisamos garantir que as 32 primeiras posições da FAT são relativas ao agrupamento da FAT, que corresponde ao numero 3. Então é feita a leitura do diretório também em um buffer auxiliar para este.

São recuperados do disco os 32 agrupamentos(256 blocos) da FAT e feitas as atualizações necessárias para o arquivo, marcar todas as posições como inexistente.

1. **int fs\_format()**

Esta função é responsável por formatar o sistema de arquivos, ou seja, aplicar todas as atualizações necessárias na estrutura para demonstrar todos os blocos como vazios e inicializar a FAT com as marcações corretas. Marca todos os diretórios como não usados e as posições da FAT como o padrão exige, as 32 iniciais como seu devido valor, e as demais como livres, ou seja, iguais a 1.

1. **int fs\_free()**

Esta função é responsável por calcular e retornar o espaço livre em disco. Para isso, ela percorre a FAT verificando as posições que podem receber valores, ou seja, estão vazias, acumulando numa variável o somatório dos bytes livres, 4096 bytes para cada posição de FAT disponível.

1. **int fs\_list(char \*buffer, int size)**

Esta função é responsável por colocar no buffer os arquivos presentes no diretório. A função varre o diretório procurando por arquivos, caso encontre os adiciona no buffer com o seguinte formato um arquivo por linha, seguido de seu tamanho e separado por dois *TABS*.

1. **int fs\_create(char\* file\_name)**

Esta função é responsável por criar um novo arquivo no sistema. Antes da criação do arquivo, a função faz duas verificações, primeira para verificar se o sistema está formatado, segunda para verificar se o sistema não está cheio ou se arquivo já existe no sistema, isto é, varre o diretório procurando por um espaço não usado e por um arquivo com o mesmo nome do arquivo que está sendo criado. A verificação de formatação é feita pela variável global *formatado* que indicará 0 caso a FAT não esteja formatada, abortando a criação. A segunda verificação procura por espaço livre no diretório e guarda em um auxiliar a posição vazia encontrada, caso haja, então prossegue para atualizar os valores da estrutura do arquivo para incluir no sistema corretamente o novo arquivo criado.

1. **int fs\_read(char \*buffer, int size, int file)**

Esta função é responsável por ler os bytes presentes no arquivo e armazená-lo em um buffer passado. Antes de ler o arquivo é necessário verificar se o disco está formatado e se o arquivo existe e se encontra aberto para leitura, então o arquivo pode ser lido usando função de leitura de blocos, e então armazenado no buffer. A variável *formatado* é verificada para a formatação e a estrutura de arquivos para procurar pelo arquivo e caso encontre, verificar se este pode ser lido, ou seja, aberto = 0.

1. **int fs\_remove(char \*file\_name)**

Esta função é responsável por apagar o arquivo passado como parâmetro, caso este esteja armazenado no sistema de arquivos. O procedimento para essa função é verificar se a FAT está formatada, em caso positivo, então irá procurar pelo arquivo no diretório, caso encontre faz as modificações na estrutura para indicar que aquele arquivo não mais existe, ou seja, seta *used* como 0, marca o arquivo inexistente = -2, zera o tamanho do arquivo, ajusta as posições do bloco como nula e marca as posições da FAT como disponíveis e então chama função para escrever a FAT e retorna 1 para arquivo removido com sucesso.

1. **int fs\_open(char \*file\_name, int mode)**

Esta função é responsável por abrir um arquivo para leitura ou escrita. Para tal execução o arquivo passado como parâmetro é procurado no diretório, mas como há duas possibilidades de manipulação do arquivo, escrita e leitura, há alguns testes a serem feitos. Para leitura, FS\_R, caso o arquivo não exista, a função retorna -1, caso exista as modificações no arquivo para marca-lo como aberto para leitura é feita. Para o caso de escrita, FS\_W, é preciso tratar diferente quando o arquivo não existe e existe, caso não seja encontrado este então será criado, e as atualizações para mostrar arquivo como aberto para escrita serão feitas, caso exista apenas atualiza as variáveis do arquivo.

1. **int fs\_close(int file)**

Esta função é responsável por fechar um arquivo, com o índice passado como parâmetro a função procura pela existência do arquivo, caso o arquivo realmente exista, é preciso retirá-lo do buffer, para isso é utilizado uma função auxiliar ***int* *descarregaBuffer(int file)*** que atualizará o buffer, removendo o referido arquivo.

1. **int fs\_write(char \*buffer, int size, int file)**

**Dificuldades**

**Conclusão**