**Pontifícia Universidade Católica de Minas Gerais**
**Instituto de Ciências Exatas e Informática**
**Unidade Educacional Praça da Liberdade**
**Bacharelado em Engenharia de Software**
**Disciplina: Sistemas Operacionais**

---

Atores:
Gabriel Henrique Souza Haddad Campos
Talita Arantes Melo

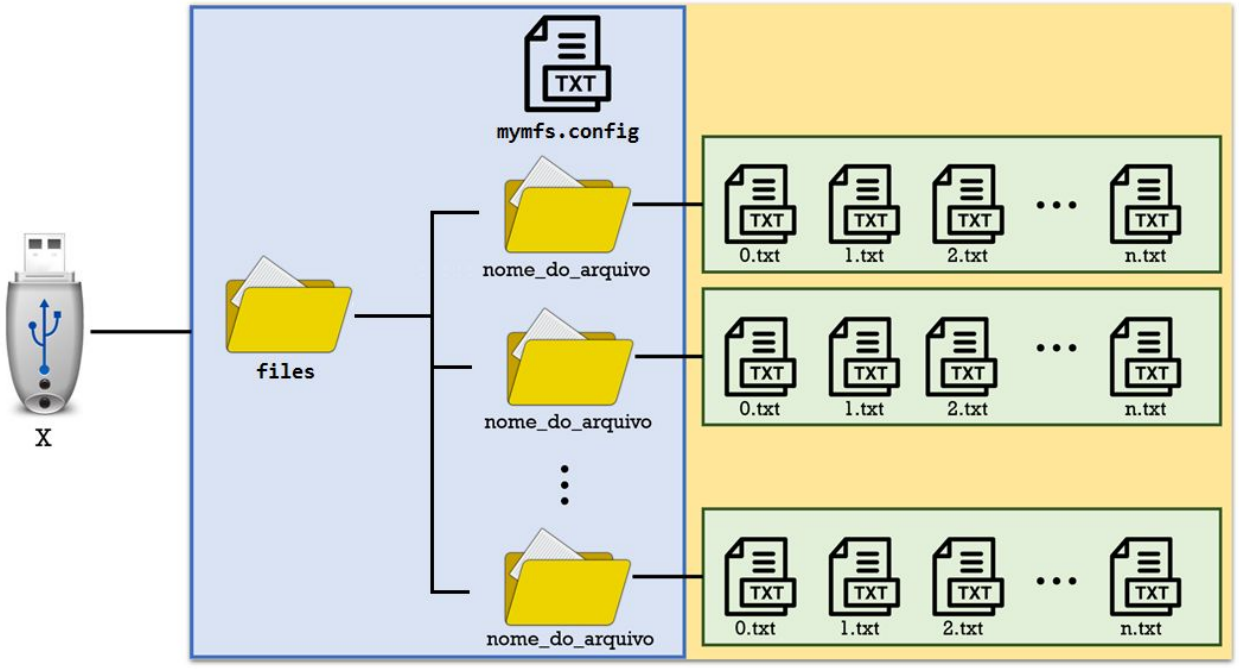
# *Trabalho Prático – MY Micro File System (mymfs) - Aquecimento*

***Obs.: O arquivo executável do mymfs é gerado ao compilar o código do arquivo mymfs.cpp***

***Compilar através do Visual Studio e o C++17***

## 1. ESTRUTURAÇÃO DAS IMPLEMENTAÇÕES

Para a implementação do Gerenciador de Arquivos *MY Micro File System (mymfs)*, foi considerado a utilização de diretórios para armazenar cada arquivo. A Figura a seguir representa a estrutura que o *mymfs* cria para armazenar os arquivos na unidade X.




O armazenamento de arquivos na unidade X é feito da seguinte forma:

1) É extraído o nome do arquivo importado e é criado um um diretório com o mesmo nome;
2) Calcula-se o tamanho do arquivo e encontra o número de arquivos de 500 KB ou menos que serão necessários (num);

3) É feito um split do arquivo importado. Cada arquivo recebe o nome equivalente ao índice do vetor. São criados n arquivos, o quais começam em 0 e terminam em num-1. Esses arquivos são armazenados dentro do diretório criado.
4) É armazenado no arquivo mymfs.config o nome do diretório e a quantidade de arquivos dentro dele.

A escolha desta arquitetura se deu pela facilidade na identificação e armazenamento de um arquivo no repositório X. Pode-se garantir as pré-condições determinadas no exercício para cada comando criado. Além disso, essa estrutura evita fragmentação interna e fragmentação externa, já que cada diretório possui o tamanho necessário para armazenar os arquivos de 500 KB ou menos do determinado arquivo que ele se refere. Além disso, o último arquivo do split não possui necessariamente 500 KB, e sim o tamanho que falta para completar o armazenamento. É considerado também o aproveitamento do espaço do arquivo config.txt, uma vez que somente 2 valores são armazenados nele. Como precisaria armazenar o nome do arquivo de qualquer forma, ele foi aproveitado para nomear o diretório onde ele se encontra dividido.

### 1.1 Comandos criados

Comando config: Assim que é executado pela primeira vez, o arquivo mymfs.config é criado, sinalizando a “instalação” do *mymfs*. Ao executar novamente, o arquivo criado é mantido.

Comando import: É criado um diretório dentro do diretório “files” (o diretório files é criado ao importar um arquivo pela primeira vez), caso não exista um, com o nome no formato “extensão-nomeArquivo” (i.e. txt-texto) e os arquivos armazenadas dentro dele são todos referentes ao arquivo importado. Ao dar ao nome dos arquivos o valor do índice do vetor, eles já são identificados conforme a ordem de apresentação, onde já se sabe qual o primeiro (0.txt) e o último (num-1).txt. Posteriormente, é escrito no arquivo mymfs.config o nome do repositório e a quantidade de arquivos dentro dele.

Comando export: Ao exportar um arquivo, é feita a procura nos diretórios com o nome do arquivo a ser exportado. Se for encontrado, no endereço passado que receberá o arquivo é feito um merge com todos os arquivos que estão dentro do diretório.

Comando listAll: Para listar os arquivos presentes no repositório, é lido o primeiro elemento de cada linha no arquivo mymfs.config. Esse elemento equivale ao nome do diretório. Para mostrar o nome no terminal, é adicionado à ele sua extensão, que também está presente no arquivo de configurações

Comando remove: Para remover o arquivo informado, primeiro verifica-se se o arquivo está presente no Mymfs. Caso esteja, o diretório com o arquivo é removido e o mymfs.config é reescrito sem a linha referente à este arquivo.

Comando removeAll: Remove o diretório, o qual contém todos os diretórios com os arquivos presentes no Mymfs. Além disso, remove todos os itens presentes no mymfs.config.

Comando grep: É feita a procura nos diretórios com o nome do arquivo a ser pesquisado. Se for encontrado, a função lê linha por linha e verifica se a palavra está presente na linha. Se estiver, é retornado o número da linha. Caso acontecer de haver uma linha quebrada em dois arquivos, a função concatena as linhas e busca pela palavra na linha concatenada.

Comando head100: É feita a procura nos diretórios com o nome do arquivo a ser pesquisado. Se for encontrado, a função lê linha por linha do começo dos arquivos e escreve na tela até haver 100 linhas escritas. Caso acontecer de haver uma linha quebrada em dois arquivos, a função concatena as linhas e escreve a nova linha na tela.

Comando tail100: É feita a procura nos diretórios com o nome do arquivo a ser pesquisado. Se for encontrado, a função conta as 100 últimas linhas do arquivo e, logo após, começa a ler linha por linha e escrever na tela.

## 2. BIBLIOTECAS PESQUISADAS

A aplicação do trabalho prático baseou-se principalmente nas bibliotecas <ifstream> e <ofstream>, responsáveis por facilitar a manipulação de arquivos e verificação da existência deles.

Outra biblioteca essencial foi a <string>, que permitiu a utilização do tipo String ao invés de vetores de caracteres (char [ ]), facilitando o desenvolvimento.

A biblioteca <math.h> foi utilizada para realizar cálculos e garantir que os arquivos respeitassem o limite de 500KB na unidade configurada com o Mymfs.

Utilizou-se a biblioteca <filesystem> (disponível apenas no C++17) para permitir a criação e exclusão de diretórios, verificações de existência de arquivos e diretórios e verificações a respeito do tamanho de arquivos.

A biblioteca <windows.h> foi utilizada no aquecimento para criação de diretórios, mas foi substituída pela <filesystem> posteriormente.

## 3. MÉTODOS UTILIZADOS DAS BIBLIOTECAS PESQUISADAS

1. **ifstream**
   a. good() - Retorna um booleano informando se o arquivo da instância ifstream existe. (Utilizado apenas no aquecimento)
   b. seekg(int posicao) - Define a posição do próximo caractere a ser extraído pela input stream.
   c. tellg() - Retorna a posição do caractere atual na input stream.
   d. read(char* s, streamsize n) - Lê os caracteres do arquivo de acordo com os parâmetros informados
   e. open() - Abre o arquivo
   f. close() - Fecha o arquivo
2. **ofstream**
   a. good() - Retorna um booleano informando se o arquivo da instância ifstream existe. (Utilizado apenas no aquecimento)
   b. write(char* s, streamsize n) - Escreve os caracteres no arquivo de acordo com os parâmetros informados
   c. open() - Abre o arquivo.
   d. close() - Fecha o arquivo
3. **math.h**
   a. ceil(double x) - Retorna o valor inteiro mais próximo ou igual ao parâmetro X
4. **string**
   a. substr(size_t pos = 0, size_t len = npos) - Retorna uma nova string, a qual particiona a string original a partir da posição e tamanho informados nos parâmetros
   b. empty() - Verifica se a string está vazia. Retorna um booleano.
   c. c_str() - Converte a string para char *.
   d. stoi() - Converte uma string para inteiro.
   e. to_string(valor) - Converte o valor do parâmetro para string.

f. find(valor) - Retorna a posição na string em que o valor do parâmetro existe.
g. strcmp(char *, char *) - Compara duas cadeias de caracteres, retorna zero se forem iguais.
h. strcpy(char *, char *) - Copia os caracteres do 2º parâmetro para o 1º (Utilizado apenas no aquecimento)

5. **windows.h (Utilizada apenas no aquecimento)**
    a. CreateDirectory (char * caminhoDiretorio, informacoesDeSeguranca) - Cria um diretório no caminho especificado. Retorna zero caso ocorra algum erro. Caso crie com sucesso, retorna um valor diferente de zero.
6. **filesystem**
    a. exists(string caminho) - Verifica se o diretório ou arquivo informado no caminho existe. Retorna um bool.
    b. create_directory (string caminho) - Cria o diretório informado no caminho, caso não exista. Retorna 0 caso não tenha criado e 1 caso tenha criado.
    c. remove_all (string caminho) - Remove todos diretórios e arquivos presentes no caminho informado.

## 4. DIFICULDADES ENCONTRADAS

As maiores dificuldades encontradas foram relacionadas a criação de diretórios. A utilização da biblioteca <*filesystem*> mostrou-se trabalhosa devido a incompatibilidade da versão do compilador GCC em nossos computadores e demandou tempo. Ao atualizar o compilador, a biblioteca continuou não sendo reconhecida, assim optamos inicial pela biblioteca <*windows.h*> durante o **aquecimento**.

Após a utilização da IDE Visual Studio 2017/2019 com o C++17, conseguimos utilizar a biblioteca <*filesystem*> e seus métodos normalmente, gerando o arquivo executável. Entretanto, tivemos muita dificuldade ao executar a compilação do **mymfs.cpp** através do terminal para criação do executável. Assim, o executável só é gerado através do Visual Studio.

Outra dificuldade foi a construção do algoritmo de separação do arquivo a ser importado em arquivos menores de 500KB e a concatenação destes no arquivo novamente. A solução foi trabalhosa e necessitou de muita pesquisa.

## 5. ANÁLISES REALIZADAS

As análises consistiram em recriar as pré-condições e validar se as pós-condições foram atendidas de acordo com a especificação do trabalho. Além disso, criamos cenários em que as pré-condições não eram atendidas, assim, o comando do Mymfs não deveria ser executado.

Os testes eram realizados a cada alteração para verificarmos se não houveram impactos e se os comandos continuavam funcionando normalmente.