

**Relatório de Projeto CheckFile**

|  |  |
| --- | --- |
| Gabriel Madeira Vieira, nº2200661 |  |
|  |  |
| Diogo dos Anjos Barbeiro, nº2200687 | **Uma imagem com pessoa, parede, sorriso, pose  Descrição gerada automaticamente** |

Declaração

“Gabriel Madeira Vieira (2200661) e Diogo dos Anjos Barbeiro (2200687) declaram sob compromisso de honra que o presente trabalho (código, relatórios e afins) foi integralmente realizado por nós, sendo que as contribuições externas se encontram claramente e inequivocamente identificadas no próprio código. Mais se declara que os estudantes acima identificados não disponibilizaram o código ou partes dele a terceiros”.

Leiria, Novembro de 2021

# Funcionalidades

## Argumentos de entrada

Foi criado um grupo para que apenas um tipo de parâmetro seja aceite.

O modo file recebe uma string, pode ser introduzido várias vezes e para o aceder é usado o –f;

O modo conjunto recebe uma string é único e para o aceder é utilizado o –b;

O modo directory recebe uma string é único e para ser acedido é usado o –d.

## Sinais

Nos sinais é usada a estrutura sigaction para para instanciar a variável de controlo. O atributo sa\_sigaction aponta para a função handle\_signal que trata dos processamentos dos sinais.

Os sinais (SIGQUIT e SIGUSR1) são associados ao programa e em caso de erro o mesmo para a sua execução. O sinal SIGUSR1 é ignorado caso o modo de execução não seja o de conjunto (batch).

Para o sinal SIGQUIT é apenas apresentada uma mensagem ao utilizador e para o SIGUSR1 é apresentada uma mensagem com o tempo de início de processamento com o número do ficheiro a ser executado e o seu nome.

Nota: a variável errno é guardada no início da função e restaurada no final.

## Comum

Todas as opções funcionam em diretorias distintas da executável, desde que o caminho indicado seja o correto face á localização do executável.

Ao iniciar o programa é reservado um espaço de memória na *heap* para a matriz de ficheiros (*fila*), sendo limpa (*zerada*) logo de seguida.

Os modos de funcionamento são apenas tipos de execução que permitem validar e adicionar a um *buffer de strings* (*fila*) os caminhos dos ficheiros a serem processados posteriormente. Isto permite adicionar futuramente mais soluções de implementação sem a necessidade de conhecer a totalidade do código (descrito na linha 163 com a função add\_to\_queue(queue, queue\_counter, string)). Além disso a vantagem de uma organização assim (com a fila) é a possibilidade de se poder intercalar os diferentes modos de execução de uma só vez. Dando assim uma vantagem e flexibilidade muito maior para a execução do mesmo caso seja pretendido (não implementado).

Após a execução dos modos, a variável auxiliar dos argumentos é libertada e o programa passa a instanciar um novo processo que ficará encarregue de percorrer a fila de ficheiros armazenando-os temporariamente num ficheiro com cada ocorrência da fila escrita por linha.

De seguida através da função execlp é executado o comando file com os argumentos –mime-type para devolver o tipo de ficheiro e -d para especificar o ficheiro gerado anteriormente com o conteúdo a processar. Os canais padrão do processo filho são redirecionados para ficheiros temporários:

1. tmp\_out – Guarda o resultado do comando file.
2. tmp\_err – Guarda os erros do comando file.
3. tmp\_batch – Guarda a fila no ficheiro de input (para argumento -d)

Segue-se a apresentação de resultados em que o ficheiro temporário tmp\_out descrito acima é lido por cada linha, é feito o *recorte* do ficheiro e da extensão passando agora pela validação final que usa os vetores supported\_extensions e supported\_types para combinar as extensões com os seus tipos (*mime types*) aprensentando então uma das 3 saídas: *ok*, *mismatch* ou *error*.

Finalizando, é libertada a variável file\_queue, fechado e eliminado o ficheiro de output temporário (tmp\_out). Caso o modo debug esteja desativado os ficheiros temporários são apagados.

## Modo ficheiro – Totalmente operacional

Supondo que dois ficheiros são dados no input do programa, “ficheiro1.html” e “ficheiro2.gif”, o modo file seria chamado da seguinte forma

* ./checkfile -f ficheiro1.html -f ficheiro2.gif

Como apresentado, é possível introduzir vários ficheiros para análise numa só chamada.

No início da análise os ficheiros são verificados afim de estarem em condições de prosseguir, se alguma das condições falhar, o programa apresenta a mensagem de erro apropriada e termina o seu processamento.

Para confirmar a existência dos ficheiros é utilizada a função ‘file\_exists’ e a função is\_regular\_file para verificar se é ficheiro regular (exemplo: não é possível analisar pastas, nem *sockets*, nem *fifos*).

Finalisando, caso as validações passem é feita uma última que confirma que o seu tamanho total não ultrapassa o máximo do buffer (256 definido na macro MAX\_STRING\_SIZE).

Se os ficheiros passarem por todas as condições irão então ser adicionados à fila com a função add\_to\_queue(files\_queue, queue\_counter, file);.

## Modo conjunto (batch) – Totalmente operacional

Supondo que o ficheiro batch tem o nome de “batch.txt”, este modo seria chamado como apresentado abaixo

* ./checkfile -b batch.txt

Este modo consegue apenas ler um ficheiro por chamada.

Tem de passar por algumas verificações e quando essas verificações não são confirmadas irá ser apresentada a mensagem de erro adequada.

Duas dessas verificações são as mesmas que no modo file: o ficheiro tem de existir e tem de ser um ficheiro regular, para tal utilizamos as mesmas funções referidas no modo file.

Por fim também será averiguado se o ficheiro é da extensão correta (.txt), para isso utilizamos a função já existente ‘*strcmp*’ e a função criada ‘*file\_extension*’.

Se todas as condições forem verificadas os ficheiros irão ser adicionados a uma fila através da função ‘add\_to\_queue(files\_queue, queue\_counter, file)’.

## Modo diretoria (directory) - Totalmente operacional

Supondo que a diretoria com os ficheiros é denominada por “diretoria”, este modo seria chamado da seguinte forma (sendo a última ‘/’ opcional)

* ./checkfile -d diretoria/

Este modo só pode analisar uma diretoria de cada vez. Tal como os modos anteriores também tem de passar por algumas validações antes de começar a analisar os ficheiros dentro da diretoria desejada.

Quando a diretoria não conseguir satisfazer essas validações um erro será apresentado com a mensagem adequada.

Começa por verificar se é possível abrir a diretoria utilizando a função já existente ‘opendir’, depois entra num ciclo que irá percorrer a diretoria e analisar os ficheiros regulares até chegar ao último dentro da mesma.

Neste modo, ao contrário dos anteriores, para saber se o ficheiro é regular é utilizada a condição ‘entity->d\_type == 8’ (foi decidido não utilizar a constante DT\_REG pois não funcionava da forma pretendida, então é-se usado o valor da mesma - 8).

O nome de cada ficheiro dentro da diretoria não pode ser superior a 256 carateres (definido na constante MAX\_STRING\_SIZE) cuja verificação é feita através da função ‘strlen’ e o ficheiro deve existir, mais uma vez confirmado através da função ‘file\_exists’.

Se estas condições se verificarem então os ficheiros serão adicionados à fila ‘add\_to\_queue(files\_queue, queue\_counter, file)’.

## Ficheiros auxiliares

### Cores (colors.c e colors.h)

* void setcolor(const char \*color);

Cores (prefixo COLOR\_): BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE e RESET

### Auxiliar de ficheiros (file\_helper.c e file\_helper.h)

* int file\_exists(const char \*filename);
* int create\_file(const char \*filename, mode\_t mode);
* int trunc\_file(const char \*filename);
* int open\_file(const char \*filename, int mode);
* off\_t file\_size(int fd)
* int is\_directory(const char \*path);
* int is\_regular\_file(const char \*path);
* char \*file\_extension(const char \*file);

### Mensagens (message.c e message.h)

* void on\_debug(int type, char \*fmt, ...);
* void on\_message(int type, char \*fmt, ...);
* void on\_error(int err, int extCode, char \*fmt, ...);

Nota: Contém macros de erro e respetivos códigos.

### Estatísticas (statistics.c e statistics.h)

* void init\_statistics(statistics\_t \*statistics);

Nota: Contém estrutura de estatísticas ( statistics\_t )

### Auxiliar de ficheiros (string\_aux.c e string\_aux.h)

* void strcut(char \*buffer, const char \*string, int min, int max);
* int array\_has\_string(const char \*\*array, int size, const char \*string);
* void strtolower(char \*string);
* void add\_to\_queue(char \*\*files\_queue, int \*queue\_counter, char \*string);
* void clean\_queue(char \*\*queue);
* char \*\*queue\_new();
* void queue\_free(char \*\*queue);

Nota: As macros que limitam a *queue* e o tamanho das *strings* são definidas caso ainda não estejam quando o *.h* for chamado. (MAX\_QUEUE e MAX\_STRING\_SIZE)

# Webgrafia

<https://stackoverflow.com/questions/5309471/getting-file-extension-in-c>

<https://stackoverflow.com/questions/6970224/providing-passing-argument-to-signal-handler>

<https://stackoverflow.com/questions/2828648/how-to-pass-a-multidimensional-array-to-a-function-in-c-and-c>

<https://www.techonthenet.com/c_language/standard_library_functions/string_h/strrchr.php>

<https://www.bookofnetwork.com/c/low-level-input-output-in-c-language>

<https://www.thegeekstuff.com/2010/10/linux-error-codes/>

<https://www.geeksforgeeks.org/signals-c-language/>

<https://linuxhint.com/signal_handlers_c_programming_language/>

<https://www.techonthenet.com/c_language/standard_library_functions/string_h/strrchr.php>

<https://stackoverflow.com/questions/24472724/expression-must-be-a-pointer-to-a-complete-object-type-using-simple-pointer-arit>

<https://overiq.com/c-programming-101/pointer-to-a-structure-in-c/>

[Linux man pages online (man7.org)](https://man7.org/linux/man-pages/)

[Linux man pages (die.net)](https://linux.die.net/man/)