Politécnico de Viseu
Tecnologia e Gestão Viseu

Assignment 4 – C Programming in Linux

Sistemas Operativos

Licenciatura em Engenharia Informática

Hugo Alexandre Pereira Afonso (30032)

Mateus Valente Frias Silva (29989)

Rodrigo de Almeida Martins (30773)

Tiago Filipe Ferreira São Bento (31489)

Outubro de 2025, Viseu

Introdução

Este relatório acompanha o desenvolvimento do assignment 4, que consiste na criação de um programa em C para Linux com três funcionalidades principais, sendo estas a contagem de linhas, palavras e caracteres num ficheiro de texto, a identificação da maior palavra e a sua posição, e a implementação de redirecionamentos de I/O com o uso de **execlp()** para replicar o comportamento do comando wc no sistema.

Alínea a) - Contagem de linhas, palavras e caracteres

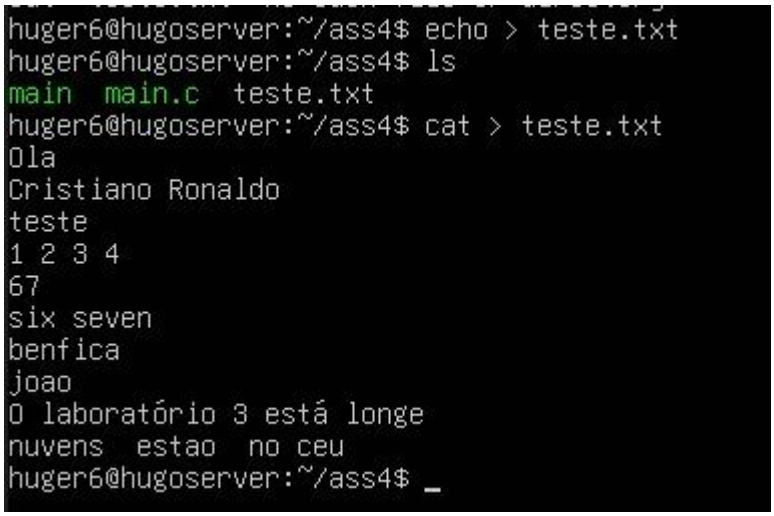
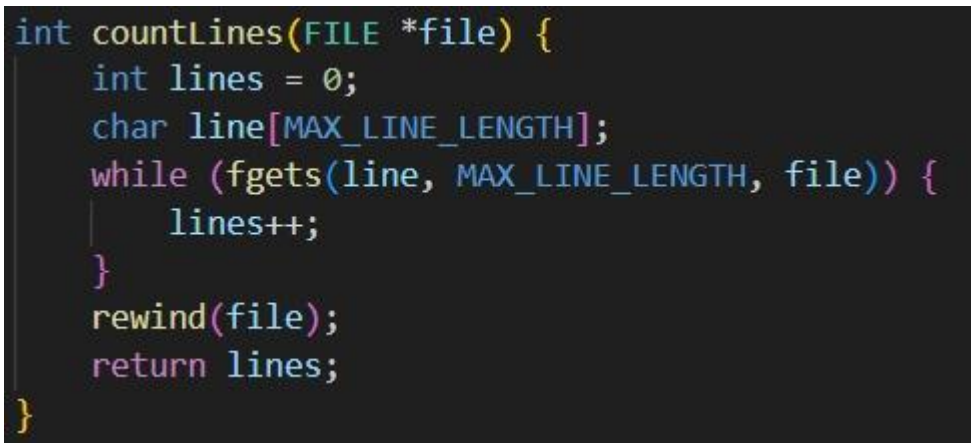


Figura 1-Preparação do Ficheiro de Texto

Nesta imagem é demonstrada a criação do ficheiro de texto que será utilizado para validar funcionamento do programa em desenvolvimento.

O conteúdo do ficheiro inclui dados variados como “Cristiano Ronaldo”, números e frases que permitirão a correta testagem das funcionalidades do programa.



```c
int countLines(FILE *file) {
    int lines = 0;
    char line[MAX_LINE_LENGTH];
    while (fgets(line, MAX_LINE_LENGTH, file)) {
        lines++;
    }
    rewind(file);
    return lines;
}
```

Figura 2-Função de contagem de linhas

Aqui é apresentado o código da função **countLines** que implementa uma contagem de linhas através da leitura sequencial do ficheiro utilizando **fgets()**. Cada chamada bem-sucedida é incrementado o contador de linhas e a função termina com **rewind(file)** para permitir operações subsequentes no ficheiro.

```c
int countChars(FILE *file) {
    int chars = 0;
    char c;
    while ((c = fgetc(file)) != EOF) {
        chars++;
    }
    rewind(file);
    return chars;
}
```

Figura 3- Função de contagem de caracteres

Foi depois feita a função **countChars** que realiza a contagem de caracteres através de uma leitura sequencial de cada carácter com o uso de **fgetc()** até que atinge o **EOF**(end of file).

```c
int countWords(FILE *file) {
    int words = 0;
    char line[MAX_LINE_LENGTH];
    int inWord = 0;
    while (fgets(line, MAX_LINE_LENGTH, file)) {
        int i = 0;
        while (line[i]) {
            if (line[i] != ' ' && line[i] != '\t' && line[i] != '\n') {
                if (!inWord) {
                    words++;
                    inWord = 1;
                }
            } else {
                inWord = 0;
            }
            i++;
        }
    }
    rewind(file);
    return words;
}
```

Figura 4-Função de contagem de palavras

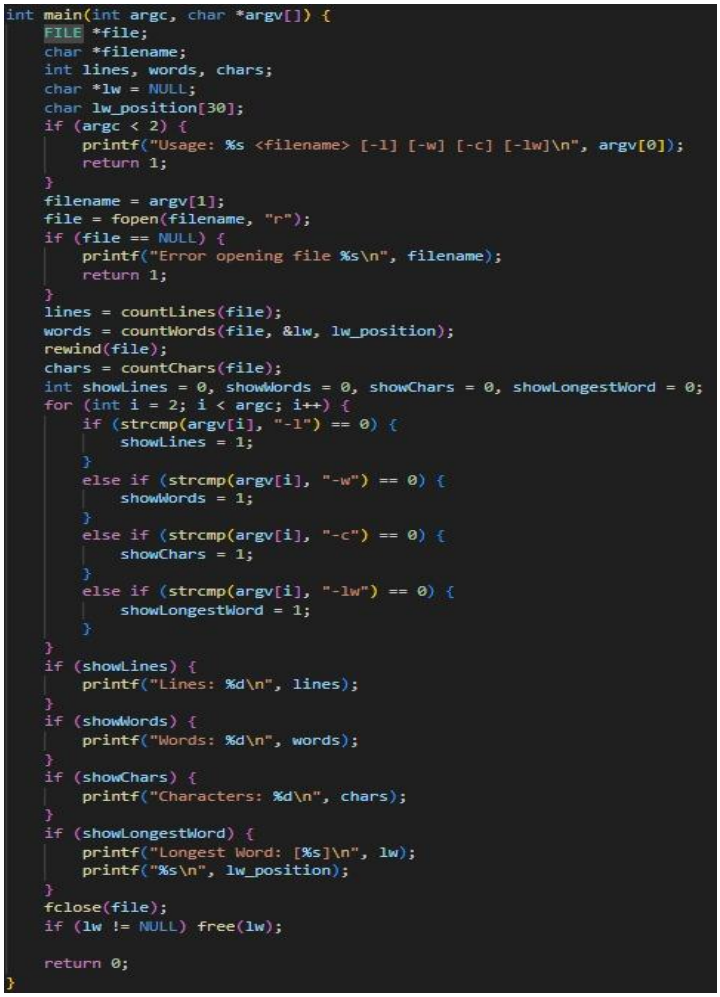
Por fim, foi feita a função **countWords** que implementa a contagem de palavras com o uso de um buffer para ler linhas e um algoritmo de máquina de estados, através da variável **inword**, para identificar transições entre espaços, incrementando o contador apenas no início de cada palavras.

```
huger6@hugoserver:~/ass4$ ./main
Usage: ./main <filename> [-l] [-w] [-c]
huger6@hugoserver:~/ass4$ ./main teste.txt -l
Lines: 10
huger6@hugoserver:~/ass4$ ./main teste.txt -w
Words: 22
huger6@hugoserver:~/ass4$ ./main teste.txt -c
Characters: 114
huger6@hugoserver:~/ass4$ _
```

Figura 5-Resultado das funções de contagem

Esta imagem mostra o resultado das funções de contagem feitas para a alínea **a)**. O programa foi testado com o ficheiro “teste.txt” e demonstrou os resultados de 10 linhas, 22 palavras e 114 caracteres.

Alínea b) – Qual a maior palavra existente no texto e em que posição em que esta se encontra na frase



```c
int main(int argc, char *argv[]) {
    FILE *file;
    char *filename;
    int lines, words, chars;
    char *lw = NULL;
    char lw_position[30];
    if (argc < 2) {
        printf("Usage: %s <filename> [-l] [-w] [-c] [-lw]\n", argv[0]);
        return 1;
    }
    filename = argv[1];
    file = fopen(filename, "r");
    if (file == NULL) {
        printf("Error opening file %s\n", filename);
        return 1;
    }
    lines = countLines(file);
    words = countWords(file, &lw, lw_position);
    rewind(file);
    chars = countChars(file);
    int showLines = 0, showWords = 0, showChars = 0, showLongestWord = 0;
    for (int i = 2; i < argc; i++) {
        if (strcmp(argv[i], "-l") == 0) {
            showLines = 1;
        }
        else if (strcmp(argv[i], "-w") == 0) {
            showWords = 1;
        }
        else if (strcmp(argv[i], "-c") == 0) {
            showChars = 1;
        }
        else if (strcmp(argv[i], "-lw") == 0) {
            showLongestWord = 1;
        }
    }
    if (showLines) {
        printf("Lines: %d\n", lines);
    }
    if (showWords) {
        printf("Words: %d\n", words);
    }
    if (showChars) {
        printf("Characters: %d\n", chars);
    }
    if (showLongestWord) {
        printf("Longest Word: [%s]\n", lw);
        printf("%s\n", lw_position);
    }
    fclose(file);
    if (lw != NULL) free(lw);

    return 0;
}
```

Figura 6-Palavra mais longa

É aqui dado o início da alínea **b)**. O código processa os argumentos da linha de comandos, abre o ficheiro e chama as funções de contagem. Implementa um sistema de flags para controlar a exibição dos resultados consoante as opções **-l, -w**, **-c** e **-lw**. Para a alínea **b),** a função **countWords** foi modificada para retornar também a palavra mais longa e sua posição.

```c
int countWords(FILE *file, char **lw, char *lw_position) {
    rewind(file);
    int words = 0;
    char line[MAX_LINE_LENGTH];
    int inWord = 0;
    size_t maxLen = 0;
    int nLines = 0;
    size_t wordLen = 0;
    int wordsPerLine = 0;

    while (fgets(line, MAX_LINE_LENGTH, file)) {
        nLines++;
        int i = 0;
        while (line[i]) {
            if (line[i] != ' ' && line[i] != '\t' && line[i] != '\n') {
                if (!inWord) {
                    words++;
                    inWord = 1;
                    wordLen++;
                }
                else {
                    wordLen++;
                }
            }
            else {
                if (inWord) {
                    if (wordLen > maxLen) {
                        maxLen = wordLen;
                        if (*lw) free(*lw);
                        *lw = strndup(&line[i - wordLen], wordLen); // only avaliable in linux
                        sprintf(lw_position, "Line: %d; Position: %d", nLines, i - wordLen);
                    }
                    inWord = 0;
                    wordLen = 0;
                }
            }
            i++;
        }
    }
    rewind(file);
    return words;
}
```

Figura 7-Modificação countWords

Como referido anteriormente a função **countWords** foi modificada para a alínea **b)**. A função agora deteta a palavra mais longa durante a contagem, usando **strndup**, que só pode ser usado em Linux, para a copiar e registando a sua posição (linha e índice). Mantém o contador de palavras e a lógica de deteção de palavras através da variável **inWord**.

```c
int countChars(FILE *file) {
    rewind(file);
    int chars = 0;
    char c;
    while ((c = fgetc(file)) != EOF) {
        chars++;
    }
    rewind(file);
    return chars;
}
```

Figura 8-Modificação da countChars

```c
int countLines(FILE *file) {
    rewind(file);
    int lines = 0;
    char line[MAX_LINE_LENGTH];
    while (fgets(line, MAX_LINE_LENGTH, file)) {
        lines++;
    }
    rewind(file);
    return lines;
}
```

Figura 9-Modificação da countLines

Na implementação da alínea **b)** foi necessária a adição do comando **rewind(file)** no início de cada função de contagem, devido a uma alteração na forma como o programa processa o ficheiro. Com a inclusão da funcionalidade de deteção da palavra mais longa e da sua localização no **countWords** foi necessário fazer esta pequena alteração nas funções de contagem. Sem o **rewind** no início da função, se o **countLines** fosse chamada primeiro, por exemplo, iria deixar o ponteiro no fim do ficheiro e assim quando a **countWords** fosse chamada ia começar a ler pelo fim do ficheiro. Porém com o **rewind** no início, é garantida a independência da ordem de chamada de cada função.

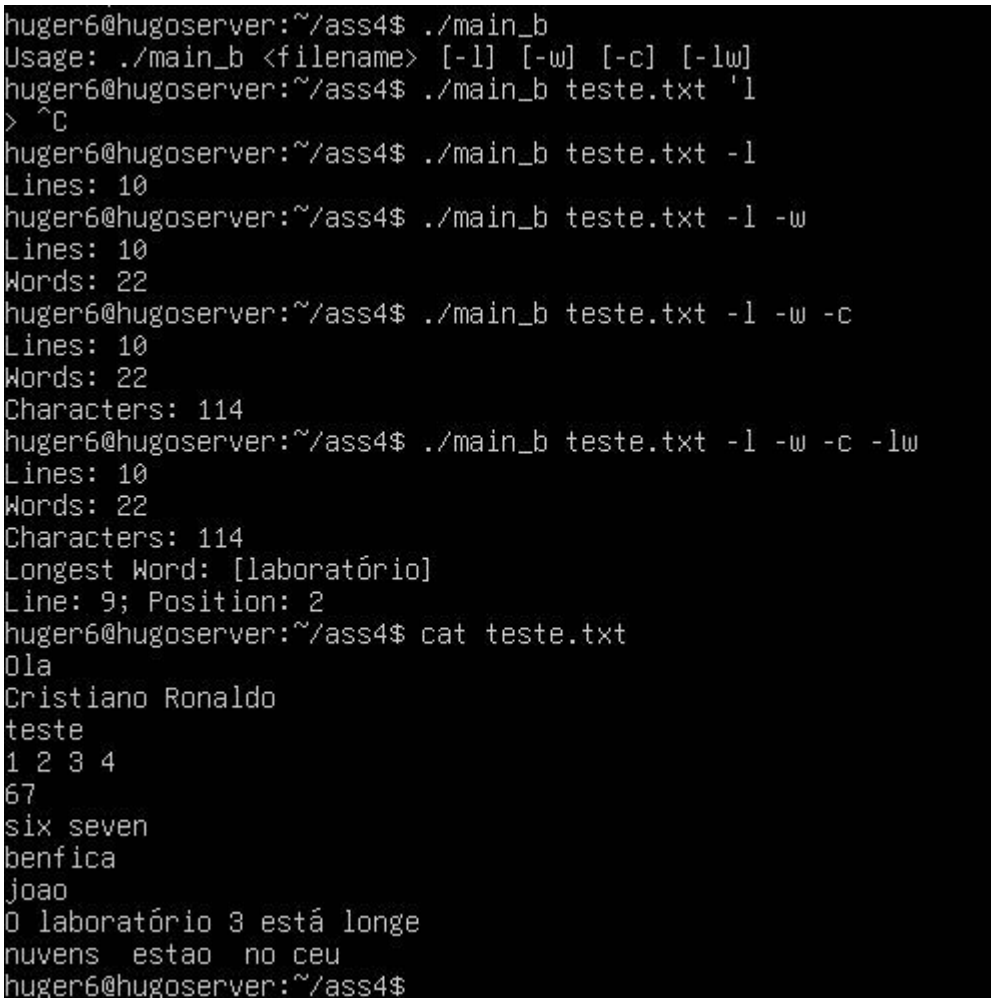
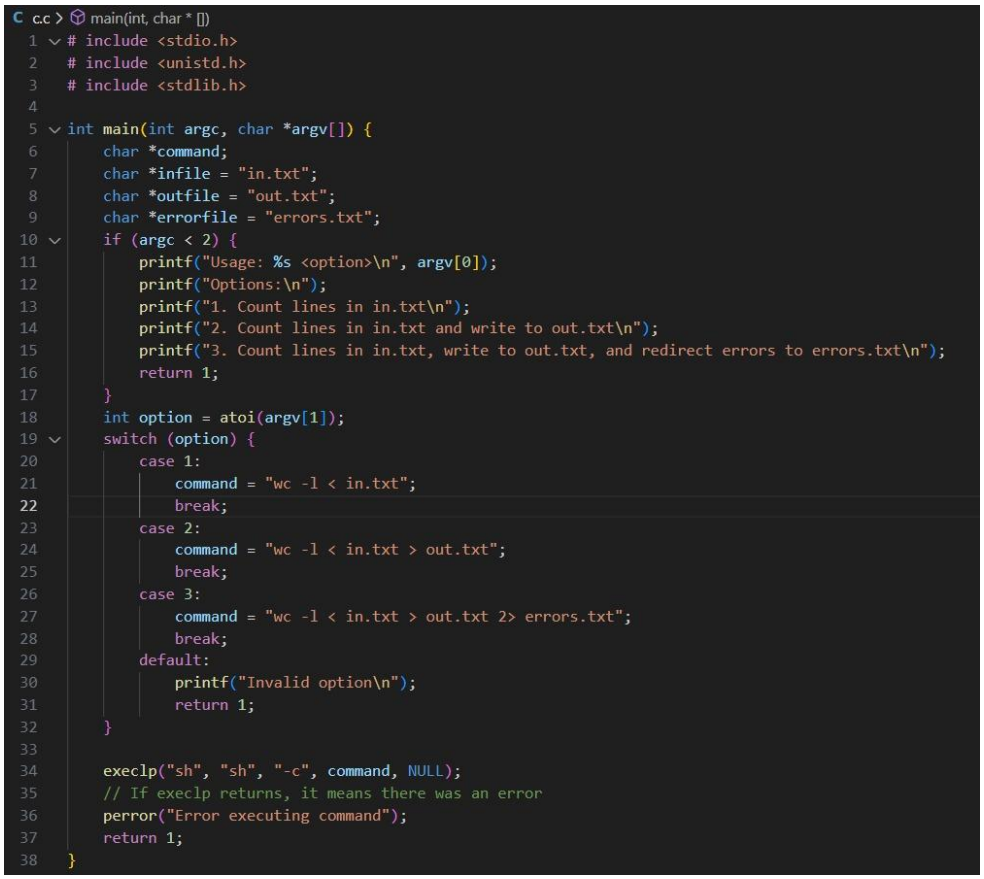


Figura 10-Deteção da maior palavra e localização da mesma

Aqui vemos o resultado do teste do programa, as opções **-l, -w** e **-c** continuam a produzir os resultados esperados de 10 linhas, 22 palavras e 114 caracteres. A nova funcionalidade de deteção da maior palavra e da localização da mesma identifica corretamente, com o uso de **-lw**, que a maior palavra é “laboratório” e que se situa na linha 9 na 2ª posição.

Alínea c) – Implementação de Redirecionamentos com base na função ‘execlp()’ e no comando ‘wc’



```c
C c.c > main(int, char * [])
# include <stdio.h>
# include <unistd.h>
# include <stdlib.h>

int main(int argc, char *argv[]) {
    char *command;
    char *infile = "in.txt";
    char *outfile = "out.txt";
    char *errorfile = "errors.txt";
    if (argc < 2) {
        printf("Usage: %s <option>\n", argv[0]);
        printf("Options:\n");
        printf("1. Count lines in in.txt\n");
        printf("2. Count lines in in.txt and write to out.txt\n");
        printf("3. Count lines in in.txt, write to out.txt, and redirect errors to errors.txt\n");
        return 1;
    }
    int option = atoi(argv[1]);
    switch (option) {
        case 1:
            command = "wc -l < in.txt";
            break;
        case 2:
            command = "wc -l < in.txt > out.txt";
            break;
        case 3:
            command = "wc -l < in.txt > out.txt 2> errors.txt";
            break;
        default:
            printf("Invalid option\n");
            return 1;
    }

    execlp("sh", "sh", "-c", command, NULL);
    // If execlp returns, it means there was an error
    perror("Error executing command");
    return 1;
}
```

Figura 11-execlp() e redireccionamentos

É aqui mostrado a implementação da alínea **c)**, que utiliza a função execlp() para executar o comando **wc** com redireccionamentos. São oferecidas 3 opções: contar as linhas do ficheiro **in.txt**, contar linhas e escrever o resultado em **out.txt**, e contar linhas, escrever em out.txt e redirecionar os erros para **errors.txt**. O código usa **execlp ("sh", "sh", "-c", command, NULL)** para executar os comandos de shell especificados. Se o **execlp** falhar, a função **perror** reporta o erro.

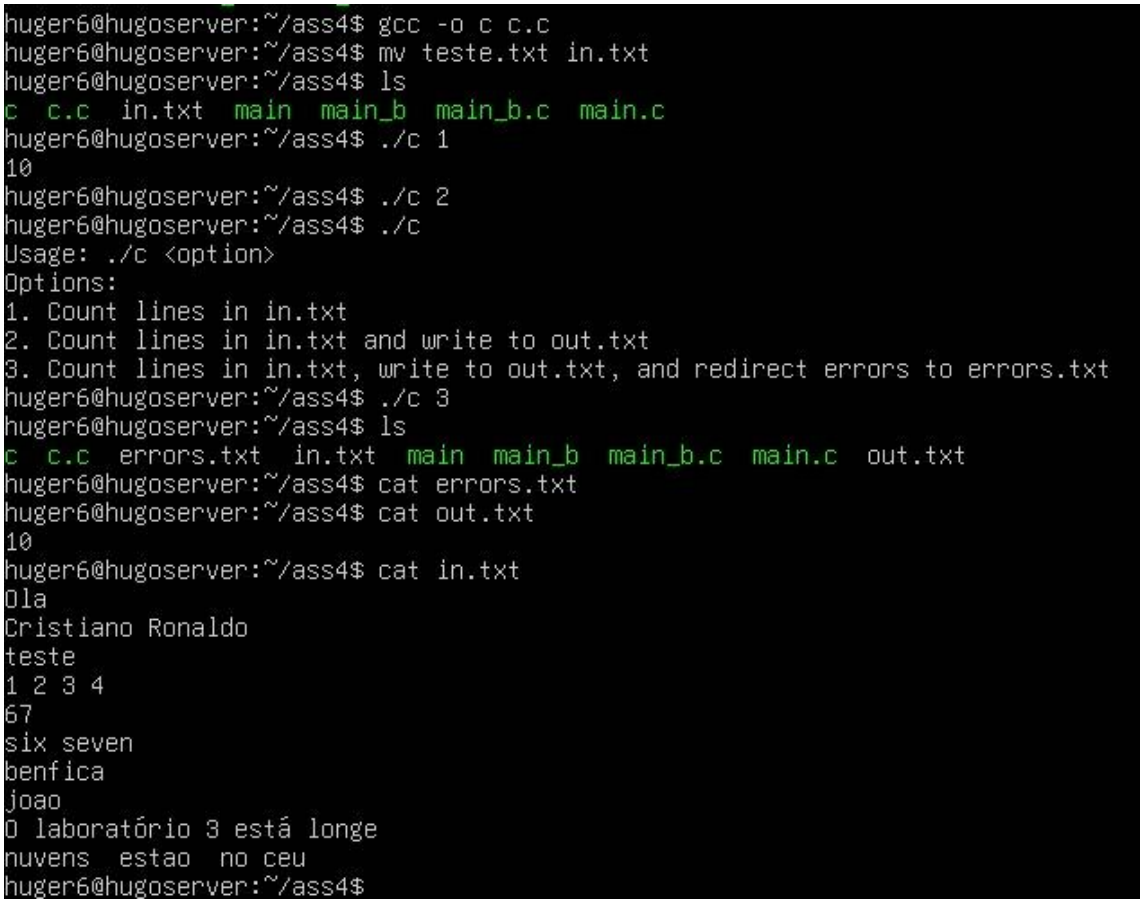


Figura 12-Teste da alínea c

O programa foi compilado como **c** e testado com as três opções. A opção 1 executou **wc -l < in.txt** diretamente no terminal, mostrando o resultado "10". As opções 2 e 3 executaram simultaneamente, e então criaram os ficheiros out.txt e errors.txt. A verificação confirmou que out.txt contém "10" (número de linhas) e errors.txt está vazio (sem erros). O conteúdo de **in.txt** mostra as 10 linhas originais, validando que todos os redirecionamentos funcionaram corretamente através da função execlp().

Conclusão

Assim, o assignment 4 foi concluído. Na alínea a), implementou-se com eficácia um contador de linhas, palavras e caracteres. A alínea b) expandiu esta funcionalidade com a deteção da palavra mais longa e sua localização, resolvendo desafios de gestão de ficheiros através do uso da função rewind(). Na alínea c), demonstrou-se a integração com o sistema operativo através da função execlp(), replicando com sucesso o comportamento do comando wc com redirecionamentos.