Politécnico de Viseu
Tecnologia e Gestão Viseu

Assignment 5 – Pipes

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)

Novembro de 2025, Viseu

Introdução

Este relatório descreve o desenvolvimento do **assignment 5**, cujo objetivo era a criação de uma aplicação multiprocesso para contagem de linhas, palavras e caracteres em múltiplos ficheiros de texto. Foram explorados mecanismos de programação, tais como a criação hierárquica de processos, a comunicação entra vários processos através de pipes, e gestão de argumentos via linha de comandos.

A aplicação implementa um sistema onde cada ficheiro é processado e organizado numa cadeia hierárquica onde cada processo cria o seguinte (P1->P2->P3…). Os resultados são comunicados ao processo pai através de pipes e agregados para a apresentação final, com suporte para redirecionamento do output para ficheiro ou consola.

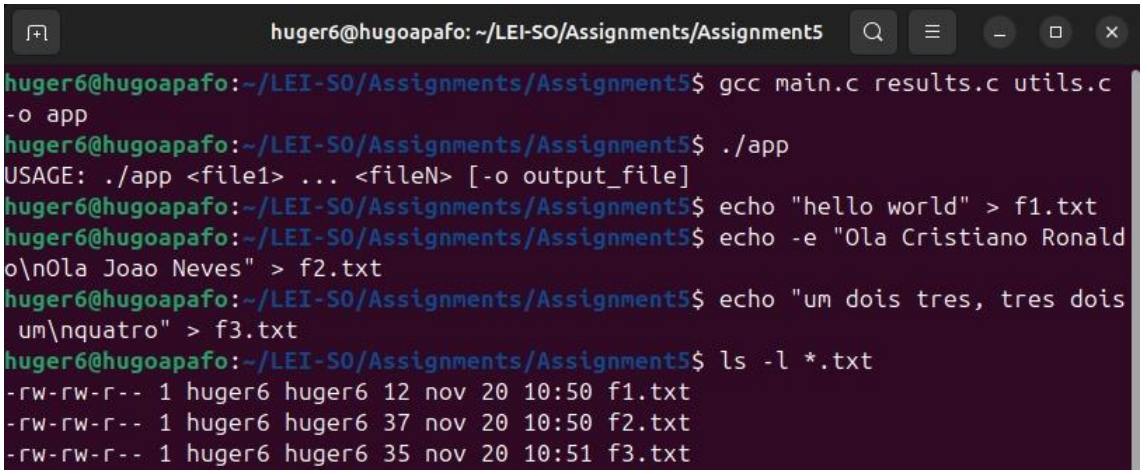

Figura 1-Compilação e preparação dos ficheiros de teste

Esta imagem mostra a fase inicial de preparação para a resolução do assigment. O programa foi compilado a partir de vários ficheiros fonte, o que gerou a “app”. Foram utilizados três ficheiros de teste (f1.txt, f2.txt e f3.txt) para validar as funcionalidades de contagem de linhas, palavras e caracteres.

```c
int main(int argc, char *argv[]) {

    if (argc < 2) {
        fprintf(stderr, "USAGE: %s <file1> ... <fileN> [-o output_file]\n", argv[0]);
        return 1;
    }

    char *output_file = NULL;
    int opt;

    // Read -o in any position
    while ((opt = getopt(argc, argv, "o:")) != -1) {
        switch (opt) {
            case 'o':
                output_file = optarg;
                break;

            default:
                fprintf(stderr, "USAGE: %s <file1> ... <fileN> [-o output_file]\n", argv[0]);
                exit(EXIT_FAILURE);
        }
    }

    // Extract files ignoring -o and its argument
    char *files[MAX_FILES];
    int file_count = 0;
```

Figura 2-Processamento de argumentos com getopt()

Esta imagem mostra a implementação do exercício 1. O código utiliza **getopt()** para tratar da opção **-o** em qualquer posição, extraindo os ficheiros de entrada enquanto ignora o switch de saída. A solução garante flexibilidade ao permitir que os argumentos sejam fornecidos em qualquer ordem.

```
58
59          // One pipe per file
60          int pipefd[MAX_FILES][2];
61
62          for (int i = 0; i < file_count; i++) {
63
64              if (pipe(pipefd[i]) == -1) {
65                  perror("pipe");
66                  exit(EXIT_FAILURE);
67              }
68
69              pid_t pid = fork();
70
71              if (pid < 0) {
72                  perror("fork");
73                  exit(EXIT_FAILURE);
74              }
75
76              if (pid == 0) {
77                  // ---------------- CHILD ----------------
78                  close(pipefd[i][READ_END]);
79
80                  Result res;
81                  memset(&res, 0, sizeof(Result));
82
83                  res.pid = getpid();
84                  res.ppid = getppid();
85                  snprintf(res.filename, sizeof(res.filename), "%s", files[i]);
```

Figura 3-Criação hierárquica de processos em cadeia

Aqui podemos ver a implementação do exercício 2 que demonstra, através do código, como é criado um processo filho opara cada ficheiro a processar através da função **fork()**. Para cada ficheiro no array, o processo atual cria um processo, o que estabelece uma relação pai-filho sequencial. Cada processo filho herda a informação do ficheiro que deve processar e preparar-se para executar a contagem.

```c
            if (pid == 0) {
                // ---------------- CHILD ----------------
                close(pipefd[i][READ_END]);

                Result res;
                memset(&res, 0, sizeof(Result));

                res.pid = getpid();
                res.ppid = getppid();
                snprintf(res.filename, sizeof(res.filename), "%s", files[i]);

                count(files[i], &res.lines, &res.words, &res.chars);

                write(pipefd[i][WRITE_END], &res, sizeof(Result));
                close(pipefd[i][WRITE_END]);

                exit(0);
            }
```

```c
void count(const char *filename, int *lines, int *words, int *chars) {
    FILE *file = fopen(filename, "r");
    if (!file) {
        fprintf(stderr, "Error opening file %s\n", filename);
        exit(EXIT_FAILURE);
    }

    int c, prev = ' ';
    *lines = *words = *chars = 0;

    while ((c = fgetc(file)) != EOF) {
        (*chars)++;
        if (c == '\n') (*lines)++;
        if (isspace(c) && !isspace(prev)) (*words)++;
        prev = c;
    }

    fclose(file);
}
```

Figura 4-contagem concorrente em processos filhos

Passando para o exercício 3, no processo filho, a função **count()** é chamada com o ficheiro atribuído, enquanto calcula, ao mesmo tempo, o número de linhas, palavras e caracteres através de uma única leitura do ficheiro. O algoritmo utiliza uma máquina de estados simples baseada no caractere anterior(**prev**) para detetar transições entre palavras.

Os resultados são depois escritos no pipe através do comando **write()**, o que permite que o processo pai receber os dados processados.

```
95              // ---------------- PARENT ----------------
96              close(pipefd[i][WRITE_END]);
97
98              Result res;
99              read(pipefd[i][READ_END], &res, sizeof(Result));
100             close(pipefd[i][READ_END]);
101
102             results[results_count++] = res;
103         }
104
105
106         // Wait for ALL children
107         while (wait(NULL) > 0);
108
109         // Print everything with the function from the second version
110         print_results(results, results_count, output_file);
111
112         return 0;
113     }
```

```
3   void print_results(Result *res, int n, const char *output_file) {
4       FILE *out = fopen(output_file, "w");
5       // File
6       if (out)
7           fprintf(out, "PID\tPPID\tFicheiro\tLinhas\tPalavras\tChars\n");
8       // Terminal
9       fprintf(stdout, "PID\tPPID\tFicheiro\tLinhas\tPalavras\tChars\n");
10
11      int lines_total = 0, words_total = 0, chars_total = 0;
12
13      // Print results
14      for (int i = 0; i < n; i++) {
15          // File
16          if (out)
17              fprintf(out, "%d\t%d\t%s\t%d\t%d\t%d\n",
18                  res[i].pid, res[i].ppid, res[i].filename,
19                  res[i].lines, res[i].words, res[i].chars);
20          // Terminal
21          fprintf(stdout, "%d\t%d\t%s\t\t%d\t%d\t\t%d\n",
22              res[i].pid, res[i].ppid, res[i].filename,
23              res[i].lines, res[i].words, res[i].chars);
24
25          // Acquire total results
26          lines_total += res[i].lines;
27          words_total += res[i].words;
28          chars_total += res[i].chars;
29      }
```

```
33      // File
34      if (out)
35          fprintf(out, "TOTAIS:\t\t\t%d\t%d\t%d\n", lines_total, words_total, chars_total);
36      // Terminal
37      fprintf(stdout, "TOTAIS:\t\t\t\t%d\t%d\t\t%d\n", lines_total, words_total, chars_total);
38
39      if (out) fclose(out);
40  }
```

Figura 5-Implementação da comunicação via pipes e agregação de resultados

Abrangindo a parte da comunicação através de pipes passamos para o exercício 4. O código faz com que os resultados sejam recebidos por cada processo filho com o uso de **read()** a partir dos pipes criados. Depois dos dados serem recebidos, o processo

armazena-os num array de estruturas e sincroniza a terminação de todos os processos filhos com o comando **wait()**.

A função **print_results()** faz a agregação dos valores torais, somando as linhas, as palavras e os caracteres de cada ficheiro processado individualmente. Enquanto isto acontece, é implementado o mecanismo de redirecionamento de output especificado, o que permite que os resultados sejam apresentados no terminal ou armazenados num ficheiro consoante a opção **-o** seja fornecida na linha de comandos.

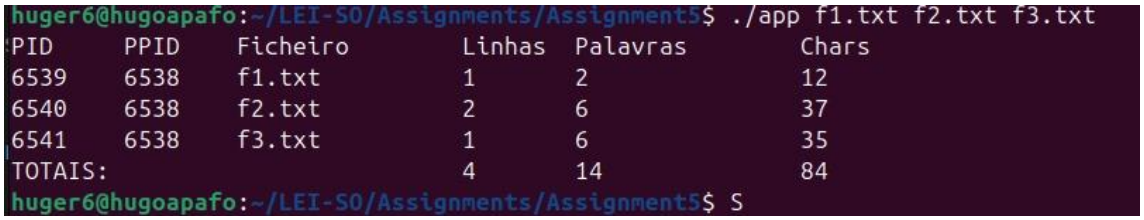

Figura 6-Apresentação dos resultados finais

Por fim são apresentados os resultados no formato especificado do código feito. O programa processou três ficheiros (f1.txt, f2.txt, f3.txt) e apresenta uma tabela clara com os PIDs, os PPIDs, os nomes dos ficheiros, as contagens de linhas, palavras e caracteres.

Conclusão

Foi implementada com sucesso uma aplicação multiprocesso em C que processa múltiplos ficheiros de texto. A aplicação utiliza pipes para comunicação entre processos, processa argumentos da linha de comandos com getopt() e apresenta os resultados formatados com totais.

Com este trabalho foi necessário utilizar os conhecimentos essenciais de sistemas operativos, incluindo a criação de processos, a comunicação interprocessos e o processamento concorrente, o que permitiu cumprir os objetivos do assignment 5.