Universidade Fernando Pessoa
Sistemas Operativos
Trabalho Prático – Parte 1

```
[375,471,499,625,657,738,758,763,795,829,857,86
5,881,972,984,986,1012,1032,1045,1059,1273,1291
,1470,1633,1634,1774,1801,1834,1851,1938,1985,2
006,2083,2119,2121,2146,2150,2158,2169,2305,233
9,2348,2451,29,43,145,164,200,235,274,333,359,3
66,376,383,446,551,600,606,621,689,719,810,864,
942,949,997,1018,1019,1091,1138,1160,1267,1326,
1339,1400,1424,1458,1466,1498,1532,1587,1609,16
38,1693,1851,1896,1931,1956,...]
```

Sistemas Operativos

Pedro Sobral
pmsobral@ufp.edu.pt

André Ribeiro Pinto
arpinto@ufp.edu.pt

Bruno Gomes
bagomes@ufp.edu.pt

Março de 2020

Universidade Fernando Pessoa

Faculdade de Ciências e Tecnologias

Objectivo:

Criar um programa que consiga ordenar um grande conjunto de dados com recurso a computação paralela.

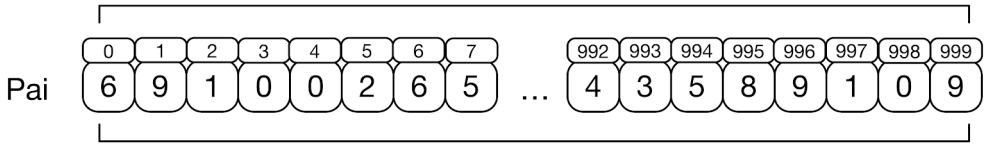
Notas:

- Este trabalho irá ser realizado individualmente ou em grupos de dois alunos. O trabalho (README, e pasta do projecto ***CodeBlocks***) tem que ser submetido até à data indicada no sistema de elearning (trabalhos) e será apresentado na aula em data a designar pelo docente.

Primeira fase:

Deve ser implementado um programa que receba por ****argv[]*** um caminho para um ficheiro onde com um grande volume de dados a ordenar. O programa deve implementar o algoritmo de ordenação ***MergeSort***.

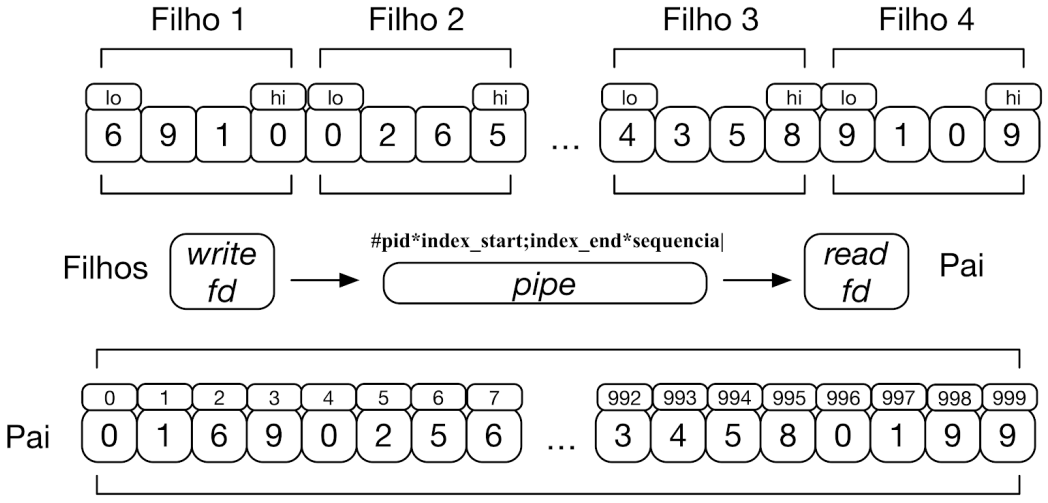
O programa deve ler e armazenar em memória o ***input*** de valores lidos do ficheiro.



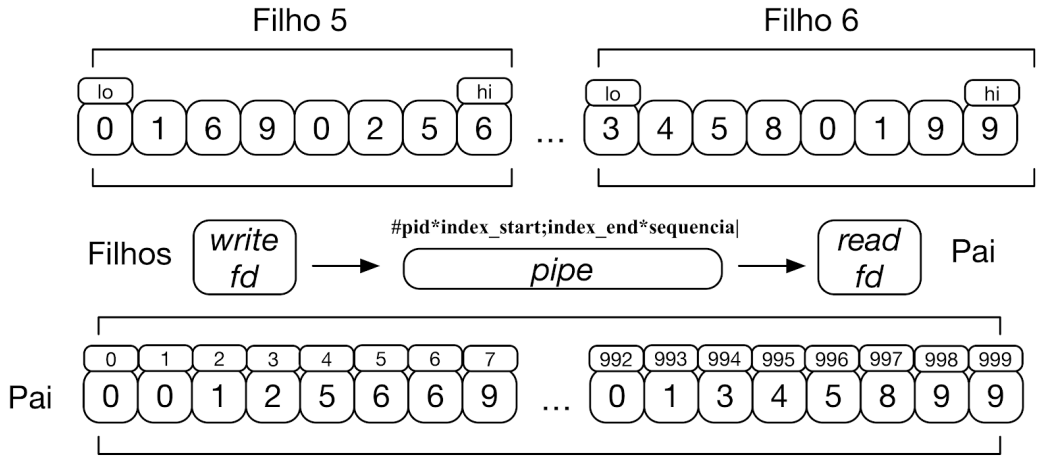
Na fase de ordenação, o programa deve seguir uma filosofia de divisão e conquista, para isso, divide o conjunto de valores lidos em ***N*** blocos. Com recurso à chamada ao sistema ***fork(), o programa deve criar N*** filhos, cada um responsável por ordenar um dos blocos, em intervalos previamente definidos pelo pai. Na primeira fase, os processos filho devem devolver as subsequências ordenadas ao processo pai recorrendo a um protocolo de mensagens indicado no exemplo seguinte:

#pid*index_start;index_end*sequência|

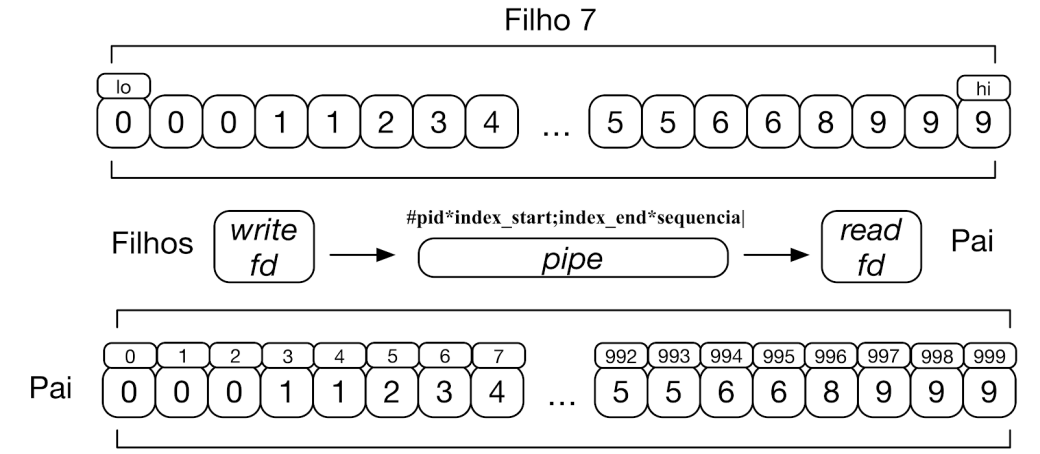
- **pid**: identificador do processo filho.
- **index_start**: limite inferior do intervalo a ordenar.
- **index_end**: limite superior do intervalo a ordenar.
- **sequência:** sequência numérica ordenados a retornar ao pai.



A fase de ordenação anteriormente descrita deve ser executada as vezes necessárias até que o conjunto de valores de encontre ordenado.



Finalmente o pai cria o último filho responsável por ordenar os dois últimos intervalos e retornar a sequência ordenada ao pai.



Após a terminação do último filho o pai grava a sequência ordenada em ficheiro.

1. (10%) Ler e armazenar o conjunto de valores a ordenar **.**

2. (25%) Criar N filhos, cada um deles deve ordenar a sua subsequência. Após a ordenação deve criar um ficheiro onde vai escrever a subsequência ordenada **.** No final o processo filho deve enviar um sinal ao pai ***SIGURS1***[1] a notificar que a sua subsequência já se encontra ordenada. O pai deve esperar pela finalização dos seus filhos.

3. (35%) Esta etapa implica que o programa permita suportar a comunicação entre processos com recurso a ***pipes***. Cada filho deve retornar a sua subsequência ordenada para o pai através do ***pipe***. O programa deve suportar um protocolo de comunicação que permita ao pai saber que filho é que ordenou determinada subsequência e a que intervalo pertence. O pai recebe as subsequências ordenadas e guarda as subsequências no array original. Nesta etapa o programa deve fazer recurso da função ***readn*** e ***writen***[2]
    a. Protocolo: #pid*index_start;index_end*sequencia|

4. (30%) Esta etapa implica que o programa permita suportar a comunicação entre processos com recurso a ***Unix Domin Sokects***[3]. Cada filho deve estabelecer conexão com o server (pai). O pai deve atender as conexões e armazenar as subsequências ordenadas array original.

Bibliografia:

[1] ***Advanced Programming in the UNIX® Environment*** - Signal - 10.14 sigaction Function
[2] ***Advanced Programming in the UNIX® Environment*** - Advanced I/O - 14.7 readn and writen Functions
[3] ***Advanced Programming in the UNIX® Environment*** - Advanced IPC - 17.2 UNIX Domain Sockets