

# **Escola Superior de Tecnologia e Gestão**

## **Licenciatura em Engenharia Informática**

### **Sistemas Operativos**

**Ano Letivo 2020/21**

### **Trabalhos Laboratoriais de Programação**

**Elaborado em: 06/05/2021**

**Daniel Santos | Nº2019133865**

**Rui Marques | Nº2019134520**

## **Índice**

<b>List of Figures.....</b>	<b>ii</b>
<b>1 Introdução.....</b>	<b>1</b>
<b>2 Processos.....</b>	<b>2</b>
2.1 Processos – Parte Teórica.....	2
2.2 Processos – Parte Prática.....	6
<b>3 Criação de Processos.....</b>	<b>10</b>
<b>4 Sinais.....</b>	<b>15</b>
<b>5 Conclusão.....</b>	<b>18</b>
5.1 Strength.....	18
5.2 Weaknesses.....	18
5.3 Opportunities.....	18
5.4 Threats.....	18
<b>6 Referências.....</b>	<b>19</b>

## List of Figures

FIGURA 1 - RESULTADO DO COMANDO PS AUX.....	3
FIGURA 2 - OBSERVAR PROCESSO FIREFOX.....	4
FIGURA 3 - PRIORIDADE DOS PROCESSOS DA APLICAÇÃO FIREFOX.....	5
FIGURA 4 - ALTERAÇÃO DA PRIORIDADE DO FIREFOX.....	5
FIGURA 5 - MENU PRINCIPAL.....	6
FIGURA 6 - MENU GESTAOPROCESSOS().....	7
FIGURA 7 - CÁLCULO DA MEDIANA E DESVIO PADRÃO COM O COMANDO "AWK"....	8
FIGURA 8 - OBTENÇÃO DO PID DO PROCESSO ATUAL.....	8
FIGURA 9 - OBTENÇÃO DA MEDIANA E DESVIO PADRÃO DE UM DETERMINADO FICHEIRO.....	9
FIGURA 10 - MENU CRIACAOPROCESSOS().....	10
FIGURA 11 - CALCULAMEDIANA INTEGRADO COM FORK();.....	10
FIGURA 12 - ESCREVE FICHEIROSUCESSO.....	11
FIGURA 13 - FILHO2 A MOSTRAR CONTEÚDO.....	11
FIGURA 14 - CÓDIGO APAGA FICHEIROS E TERMINA PROCESSOS FILHO.....	12
FIGURA 15 - BIBLIOTECAS E DECLARAÇÃO GLOBAL DOS PROCESSOS FILHOS.....	13
FIGURA 16 - GNUPLOT.....	13
FIGURA 17 - FICHEIROS ACABADOS DE CRIAR (OS DOIS PRIMEIROS).....	14
FIGURA 18 - APAGAR FICHEIROS DE RESULTADOS.....	14
FIGURA 19 - SUBMENU FICHA 03.....	15
FIGURA 20 - PROCESSO PAI E PROCESSOS FILHOS.....	15
FIGURA 21 - FUNÇÕES USADAS.....	16
FIGURA 22 - FUNÇÃO CALCULOS PARA O CALCULO DA MEDIANA, MAXIMO E MINIMO.....	17
FIGURA 23 - MENSAGEM DE RECEÇÃO DO SINAL.....	17

## **1 Introdução**

No âmbito da disciplina de Sistemas Operativos foi pedido para fazer a realização de 3 fichas abordando vários conceitos, como a explicação e programação de processos, a sua criação e sinais. Vais ser apresentado a resolução dos vários exercícios, sempre apresentando as respetivas capturas de ecrã para comprovar os vários resultados obtidos.

Tomámos a iniciativa de explicar primeiro todos os conceitos da ficha e depois apresentar o código com os seus comentários devidos.

O trabalho foi realizado no Fedora usando uma máquina virtual.

## **2 Processos**

### **2.1 Processos - Parte Teórica**

Podemos exibir informações detalhadas sobre os processos listados no diretório /proc usando comandos de processo. O diretório /proc também é conhecido como sistema de arquivos do processo (PROCFS). As imagens dos processos ativos são armazenadas no PROCFS pelo número de identificação de processo.

Os comandos de processo a seguir mostram detalhes sobre um processo no diretório /proc. Os comandos `prun` e `pstop` iniciam e interrompem um processo:

`pcred` - Exibe informações de credencial do processo

`pfiles` - Relata informações `fstat` e `fcntl` para arquivos abertos em um processo

`pflags` - Exibe sinalizadores de rastreamento / `proc`, sinais pendentes e retidos e outras informações de status

`pldd` - Lista as bibliotecas dinâmicas que estão vinculadas a um processo

`pmap` - Exibe o mapa do espaço de endereço de cada processo

`prun` - Inicia cada processo

`psig` Lista as ações de sinal e manipuladores de cada processo

`pstack` - Exibe um rastreamento de pilha hexadecimal + simbólico para cada processo leve em cada processo

`pstop` - Para cada processo

`ptime` - Cronometra um processo usando contabilidade de microestado

`ptree` - Mostra as árvores de processo que contêm o processo

**pswait** - Exibe informações de status após o término de um processo

**pwdx** - Mostra o diretório de trabalho atual para um processo

O comando **ps** permite que verifique o status dos processos ativos em um sistema e também exibe informações técnicas sobre os processos. Esses dados são úteis para tarefas administrativas, como determinar como definir prioridades de processo.

**NI** - O bom número do processo, o que contribui para sua prioridade de escalonamento. Tornar um processo "mais agradável" significa diminuir sua prioridade.

**PID** - O ID do processo.

**PPID** - O ID do processo pai.

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.1	0.4	110852	16892	?	Ss	21:12	0:02	/usr/lib/syst
root	2	0.0	0.0	0	0	?	S	21:12	0:00	[kthreadd]
root	3	0.0	0.0	0	0	?	I<	21:12	0:00	[rcu_gp]
root	4	0.0	0.0	0	0	?	I<	21:12	0:00	[rcu_par_gp]
root	5	0.0	0.0	0	0	?	I	21:12	0:00	[kworker/0:0-
root	6	0.0	0.0	0	0	?	I<	21:12	0:00	[kworker/0:0H
root	9	0.0	0.0	0	0	?	I<	21:12	0:00	[mm_percpu_wq
root	10	0.0	0.0	0	0	?	S	21:12	0:00	[rcu_tasks_kt
root	11	0.0	0.0	0	0	?	S	21:12	0:00	[rcu_tasks_ru
root	12	0.0	0.0	0	0	?	S	21:12	0:00	[rcu_tasks_tr
root	13	0.0	0.0	0	0	?	S	21:12	0:00	[ksoftirqd/0]
root	14	0.0	0.0	0	0	?	I	21:12	0:00	[rcu_sched]
root	15	0.0	0.0	0	0	?	S	21:12	0:00	[migration/0]
root	16	0.0	0.0	0	0	?	S	21:12	0:00	[cpuhp/0]
root	17	0.0	0.0	0	0	?	S	21:12	0:00	[kdevtmpfs]
root	18	0.0	0.0	0	0	?	I<	21:12	0:00	[netns]
root	19	0.0	0.0	0	0	?	I<	21:12	0:00	[inet_frag_wq
root	20	0.0	0.0	0	0	?	S	21:12	0:00	[kauditd]
root	21	0.0	0.0	0	0	?	S	21:12	0:00	[oom_reaper]

Figura 1 - Resultado do comando **ps aux**

1ª coluna - Indica a que utilizador pertence.

2ª coluna - Mostra o ID do processo

- 3º coluna - Indica o consumo que o processo tem no cpu
- 4º coluna - Proporção do tamanho do processo para a memória física na máquina
- 5º coluna - Uso de memória virtual de todo o processo
- 6º coluna - Indica a memória física não trocada que uma tarefa usou.
- 7º coluna - Indica se controla terminal
- 8º coluna - Estado de processo de vários caracteres
- 9º coluna - Hora ou data de início do processo
- 10º coluna - Tempo cumulativo de CPU
- 11º coluna - Comando com todos os argumentos

O grep é um comando com uma função simples: ele procura por pedaços de texto (strings) dentro de arquivos ou diretórios e retorna arquivos/diretórios de uma string que foi encontrada.

```
[danielsantos@fedora ~]$ ps aux |grep firefox  
daniels+  3217  0.0  0.0 221412  788 pts/0    S+   22:05   0:00 grep --color=  
auto firefox
```

Figura 2 - Observar processo firefox

Depois fomos observar a prioridade dos processos Firefox.

```
[danielsantos@fedora ~]$ top | grep firefox
 4229 daniels+  20    0 2868240 264620 135900 S    2,7    6,6    0:02.90 firefox
 4229 daniels+  20    0 2868240 263428 135900 S    1,3    6,6    0:02.94 firefox
 4229 daniels+  20    0 2868240 263452 137372 S    0,7    6,6    0:02.96 firefox
 4229 daniels+  20    0 2868240 259964 137564 S    0,3    6,5    0:02.97 firefox
 4229 daniels+  20    0 2868240 260516 137564 S    0,7    6,5    0:02.99 firefox
4229 daniels+  20    0 2872616 262240 141096 R    1,7    6,5    0:03.04 firefox
```

Figura 3 - Prioridade dos processos da aplicação Firefox

Para alterar a prioridade do processo tivemos que entrar no root do sistema e usar o comando “renice” que serve para ajustar o nice do processo (a prioridade).

Quanto menor o nice maior a prioridade desse processo.

```
[root@fedora danielsantos]# renice -n -10 -p 4229
4229 (process ID) prioridade antiga 0, nova prioridade -10
[root@fedora danielsantos]# top | grep firefox
 4229 daniels+  10  -10 2860048 258232 139716 S    4,0    6,4    0:03.52 firefox
 4229 daniels+  10  -10 2860048 258296 139716 S    0,7    6,4    0:03.54 firefox
 4229 daniels+  10  -10 2860048 258352 139716 S    5,6    6,4    0:03.71 firefox
4229 daniels+  10  -10 2860048 258408 139716 R    1,7    6,4    0:03.76 firefox
 4229 daniels+  10  -10 2860048 258672 139716 S    1,0    6,4    0:03.79 firefox
```

Figura 4 - Alteração da prioridade do firefox

Passando agora ao próximo conceito, o comando “pstree” é um comando que mostra os processos em execução em forma de árvore. É usado como uma alternativa mais visual ao comando ps. A raiz da árvore é init ou o processo com o pid fornecido.

O comando “top” mostra a atividade do processador de sua máquina Linux e também exhibe tarefas gerenciadas pelo kernel



em tempo real. Ele mostra também o processador e a memória sendo usados e outras informações, como processos em execução.

O comando “system” passa o nome do comando ou nome do programa especificado pelo comando para o ambiente host para ser executado pelo processador do comando e retorna após o comando ter sido concluído.

A função “exec” substitui a imagem do processo atual por uma nova imagem do processo. A nova imagem deve ser construída a partir de um arquivo executável normal denominado novo arquivo de imagem do processo. Não haverá retorno de um “exec()” bem-sucedido, porque a imagem do processo de chamada é sobreposta pela nova imagem do processo.

## **2.2 Processos - Parte Prática**

Passando agora à parte prática da ficha, começámos por fazer o menu para todas as fichas:

```
void Signals(){
}

void main(){
    int menu=0;
    printf("Bem vindo!\n");
    do{
        printf("1.Gestao de processos  2.Criacao de processos  3.Signals 0.Sair\n");
        scanf("%d",&menu);
        switch (menu) {
            case 0:printf("Adeus!");
                return;
                break;
            case 1:GestaoProcessos();
                break;
            case 2:CriacaoProcessos();
                break;
            case 3:Signals();
                break;
            default: printf("error\n");
        }
    }while(1);
}
```

Figura 5 - Menu Principal

Consideramos a ficha 1 bastante fácil até chegarmos à opção Mediana e desvio padrão que funcionámos pela primeira vez com o comando “awk”.

Depois de chegarmos a uma “fórmula” conseguimos por a funcionar com o ficheiro PL\_Programacao\_00\_Database.

```
void GestaoProcessos(){
    int opcao=0;
    printf("Consultar PIDs:\n1.PID do processo atual  2.PID do processo pai\nVisua
    scanf("%d",&opcao);
    switch (opcao) {
        case 1:printf("PID do processo atual:%d\n",getpid());
        break;
        case 2:printf("PID do processo pai:%d\n",getppid());
        break;
        case 3:system("ps aux");
        break;
        case 4:system("pstree");
        break;
        case 5:execl("/usr/bin/top","top",NULL);
        exit(0);
        break;
        case 6:
        char nomeFicheiro[100];
        FILE *Data;
```

Figura 6 - Menu GestaoProcessos()

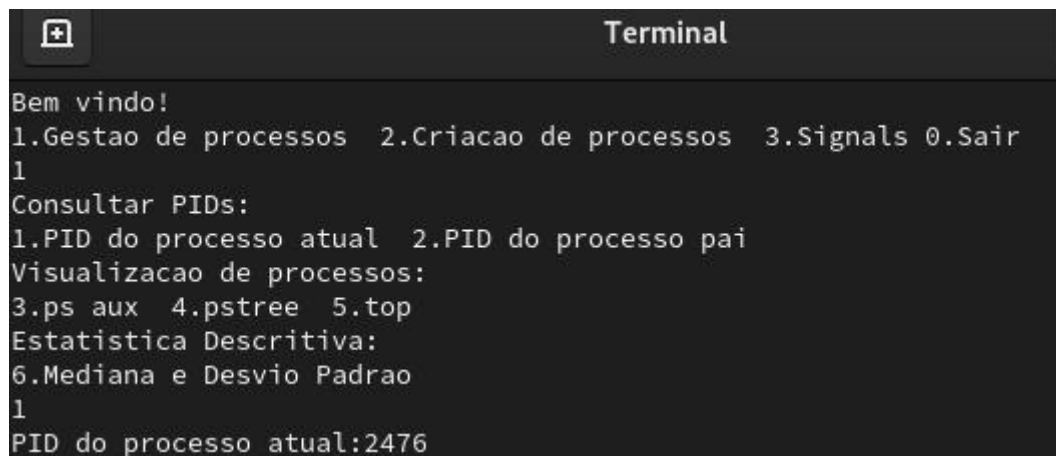
```
case 6:
char nomeFicheiro[100];
FILE *Data;

printf("Insira nome do ficheiro a fazer a mediana e o desvio padrao:\n");
fflush(stdin);
scanf("%s", nomeFicheiro);

if(Data = fopen(nomeFicheiro, "rb")){
    printf("O ficheiro inserido foi encontrado!\n");
    printf("Mediana:\n");
    system("awk -F ' ' 'NR>1 && NF{count[NR]=$4} END{if(NR%2){print count[(NR+1)/2]}}");
    printf("Desvio Padrao:\n");
    system("awk -F ' ' 'NR > 1 && NF {x[NR]=$4; soma+= $4} END{media=soma/NR; printf \"%f\", media}");
    printf("\n");
}else{
    printf("Nao foi possivel encontrar o ficheiro....\n");
}
break;
default: printf("error\n");
```

Figura 7 - Cálculo da mediana e desvio padrão com o comando "awk"

E com esta última opção apresentada só falta mostrar os testes:



```
Terminal

Bem vindo!
1.Gestao de processos  2.Criacao de processos  3.Signals 0.Sair
1
Consultar PIDs:
1.PID do processo atual  2.PID do processo pai
Visualizacao de processos:
3.ps aux  4.pstree  5.top
Estatistica Descritiva:
6.Mediana e Desvio Padrao
1
PID do processo atual:2476
```

Figura 8 - Obtenção do PID do processo atual

O submenu GestaoProcessos() apresenta o seu conteúdo dividido por secções (Consultar PIDs, Visualizacao de processos, Estatistica Descritiva).

As 5 primeiras opções são bastante simples e só apresentam resultados.

A 6ª opção (Mediana e Desvio Padrao) já é diferente pois vai perguntar pelo nome do ficheiro (neste caso usamos o ficheiro exemplo, PL\_Programacao\_00\_Database) e só depois dará a mediana e o desvio padrão:

```
Consultar PIDs:
1.PID do processo atual  2.PID do processo pai
Visualizacao de processos:
3.ps aux  4.pstree  5.top
Estatistica Descritiva:
6.Mediana e Desvio Padrao
6
Insira nome do ficheiro a fazer a mediana e o desvio padrao:
PL_Programacao_00_Database.csv
O ficheiro inserido foi encontrado!
Mediana:
26
Desvio Padrao:
78.7063

1.Gestao de processos  2.Criacao de processos  3.Signals  0.Sair
```

Figura 9 - Obtenção da mediana e desvio padrão de um determinado ficheiro

### 3 Criação de Processos

Para esta ficha, como não tem parte teórica começamos logo por fazer o menu CriacaoProcessos():

```
void CriacaoProcessos(){
    int opcao=0;
    printf("1.Calculo da mediana, maximo e minimo  2.Apagar ficheiros de resultados  0.Sair\n");
    scanf("%d",&opcao);
    switch (opcao) {
```

Figura 10 - Menu CriacaoProcessos()

```
filho1 = fork();

if(Data = fopen(nomeFicheiro,"rb")){
    if(filho1 < 0){
        printf("Erro - filho1\n");
    }else{
        if(filho1 == 0){
            //calculaMediana
            char calculaMediana[100] = "awk -F ',' 'NR>1 && NF{count[NR]=$";

            strcat(coluna, "} END{if(NR%2){print count[(NR+1)/2];}else{print (count[NR/2] + count[(NR/2)+1])/2;}}' ");
            strcat(coluna,nomeFicheiro);
            strcat(coluna, " >> ficha02_mediana.dat");
            strcat(calculaMediana,coluna);

            system(calculaMediana);
```

Figura 11 - calculaMediana integrado com fork();

Depois calculamos a mediana, máximo e mínimo e só, quando conseguimos por a funcionar é integramos com o fork();

Depois passo a escrever no ficheiroSucesso e depois de escrito passo para o filho2:

```
//escreveFicheiroSucesso
FILE * ficheiroSucesso = fopen("ficha02_sucesso.dat","a+b");
time_t tempoAtual = time(&tempoAtual);
struct tm *ponteiroTempo = gmtime(&tempoAtual);

char informacaoSucesso[] = ("Operacao realizada com sucesso. Data:");
strcat(informacaoSucesso,asctime(ponteiroTempo));

fwrite(informacaoSucesso,sizeof(informacaoSucesso),1,ficheiroSucesso);
fclose(ficheiroSucesso);

printf("A informacao foi enviada para o ficheiro ficha02_mediana.dat com sucesso. PID:%d\n",getpid());
}
}

filho2 = fork();
```

Figura 12 - Escreve ficheiroSucesso

O filho dois tem como função indicar o conteúdo do ficheiro e mostrar a função gráfica (para tal, usámos o comando gnuplot):

```
filho2 = fork();

if(filho2<0){
    printf("Erro - filho2\n");
}else{
    if(filho2 == 0){
        printf("Conteudo do ficheiro ficha02_mediana.dat:\n");
        system("cat ficha02_mediana.dat");
        printf("Conteudo do ficheiro ficha02_sucesso.dat:\n");
        system("cat ficha02_sucesso.dat");

        //Função Grafico
        system("gnuplot -e \"plot 'ficha02_mediana.dat' title 'mediana';pause mouse;\"");

        printf("A informacao do grafico foi mostrado com sucesso. PID:%d\n",getpid());
        exit(0);
    }
}
```

Figura 13 - filho2 a mostrar conteúdo

Como última opção desta ficha temos que apagar os ficheiros de resultados com ambos os filhos em execução.

O filho1 apaga o ficheiro ficha02\_mediana.dat e o filho2 apaga o ficheiro ficha02\_sucesso.dat.

Por último, depois de apagado os ficheiros, terminamos os processos filho:

```
case 2:
int estado;

filho1 = fork();
filho2 = fork();

if(filho1 < 0 && filho2 < 0){
    printf("Erro - filhos\n");
}else{
    if(filho1 == 0 && filho2 == 0){
        system("rm ficha02_mediana.dat ficha02_sucesso.dat");

        printf("Os ficheiros foram apagados com sucesso. PID:%d\n",getpid());
    }else{
        filho1 = wait(&estado);
        filho2 = wait(&estado);

        printf("Os processos filho terminaram!. PID:%d\n",getpid());
    }
}
break;
default: printf("Erro\n");
```

Figura 14 - Código apaga ficheiros e termina processos filho



Por fim, os processos filhos são declarados globalmente:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <time.h>

pid_t filho1, filho2;
```

Figura 15 - Bibliotecas e declaração global dos processos filhos

Ao compilar e aceder à 1ª opção do submenu de CriacaoProcessos() iremos obter informação por texto mas também iremos criar 2 ficheiros e um gráfico:

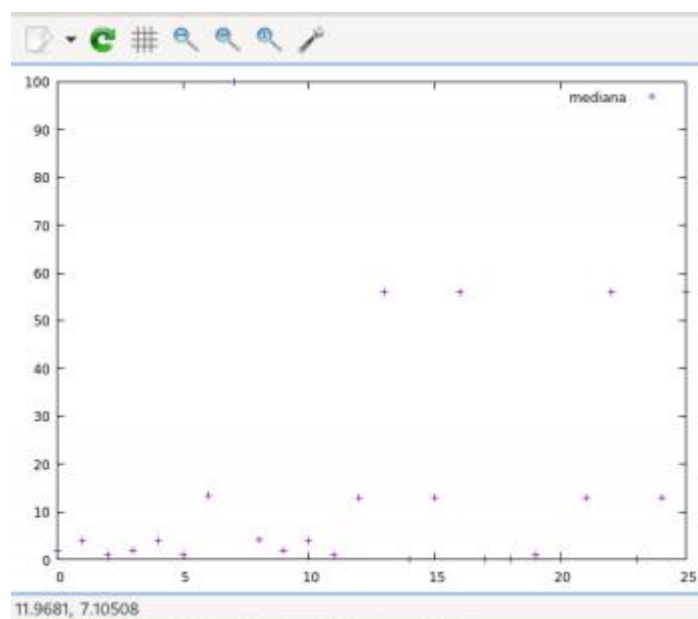


Figura 16 - Gnuplot



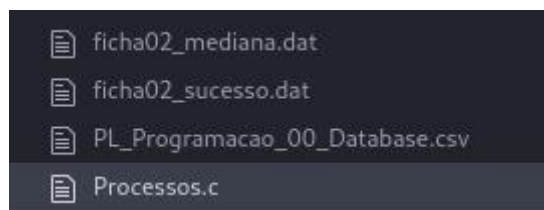


Figura 17 - Ficheiros acabados de criar (os dois primeiros)

```
1.Calculo da mediana, maximo e minimo 2.Apagar ficheiros de resultados 0.Sair  
2
```

```
Os ficheiros foram apagados com sucesso. PID:3665
```

```
Os processos filho terminaram!. PID:3664
```

Figura 18 - Apagar ficheiros de resultados

## 4 Sinais

Nesta ficha começamos por criar na função Signals o submenu:

```
void Signals(){
    int opcao=0;
    printf("1.Calculo da mediana, maximo e minimo  2.Apagar ficheiros de resultados  0.Sair\n");
    scanf("%d",&opcao);
    switch (opcao) {
        case 1: ;
```

Figura 19 - Submenu Ficha 03

Depois criamos os dois processos filho e criamos todos os envios de sinais entre processo pai e processos filhos:

```
filhoa = fork();
if(filhoa < 0){
    printf("Erro - filhoA\n");
}else{
    if(filhoa == 0){
        signal(SIGUSR1, calculos);
        signal(SIGUSR2, mostrar);
        for(;;);
    }
    else{
        filhob = fork();
        if(filhob < 0){
            printf("Erro - filhoB\n");
        }else{
            if(filhob == 0)
            {
                signal(SIGTERM, apagar);
                for(;;);
            }
            else{
                printf("Sou o processo pai (PID %d),e criei os processosA (PID %d) e B(PID %d)\n", getppid(), filhoa, filhob);
                printf("Sinal SIGTERM enviado pelo processo pai(PID %d)ao processo filhoB (PID %d)", getppid(), filhob);
                kill(filhob, SIGTERM);
                sleep(3);
            }
        }
    }
}
```

Figura 20 - Processo pai e processos filhos

O processo pai envia os sinais para os processos filhos e dependendo do sinal estes acedem as diferentes funções.

Por fim adaptámos os metodos para a execução dos requisitos, tornando-os em funções para o uso com os sinais:

```
void mostrar(){
    signal(SIGUSR2, mostrar);
    printf("Sinal SIGUSR2 recebido pelo processo com PID %d\n", getpid());
    printf("Conteudo do ficheiro ficha03_mediana.dat:\n");
    system("cat ficha03_mediana.dat");
    printf("Conteudo do ficheiro ficha03_sucesso.dat:\n");
    system("cat ficha03_sucesso.dat");

    //Função Grafico
    system("gnuplot -e \"plot 'ficha03_mediana.dat' title 'mediana';pause mouse;\"");

    printf("A informacao do grafico foi mostrado com sucesso. PID:%d\n",getpid());
    exit(0);
}
void apagar()
{
    signal(SIGTERM, apagar);
    printf("Sinal SIGTERM recebido pelo processo com PID %d\n", getpid());
    system("rm ficha03_mediana.dat ficha03_sucesso.dat");

    printf("Os ficheiros foram apagados com sucesso. PID:%d\n",getpid());
    exit(0);
}
```

Figura 21 - Funções usadas

```
void calculos()
{
    char nomeFicheiro[100];
    char linha[100];
    char coluna[100];
    FILE *Data;
    signal(SIGUSR1, calculos);
    printf("Sinal SIGUSR1 recebido pelo processo com PID %d\n", getpid());
    printf("Insira nome do ficheiro a fazer a mediana, o maximo e o minimo:\n");
    fflush(stdin);
    scanf("%s", nomeFicheiro);

    printf("Insira a linha cabecalho do ficheiro:\n");
    fflush(stdin);
    scanf("%s", linha);

    printf("Insira a coluna de dados a analisar:\n");
    fflush(stdin);
    scanf("%s", coluna);
    if(Data = fopen(nomeFicheiro, "rb")){
        char calculaMediana[100] = "awk -F ' ' 'NR>1 && NF{count[NR]=$";

        strcat(coluna, "} END{if(NR%2){print count[(NR+1)/2];}else{print (count[NR/2] + count[(NR/2)+1])/2;}}' ";
        strcat(coluna, nomeFicheiro);
        strcat(coluna, " >> ficha03_mediana.dat");
        strcat(calculaMediana, coluna);

        system(calculaMediana);

        //calculaMaximo
        char calculaMaximo[100] = ("awk -F ' ' 'BEGIN{numeroMaximo=0}{if ($");

        strcat(coluna, ">0+numeroMaximo) numeroMaximo=$");
        strcat(coluna, "} END{print numeroMaximo}' ");
        strcat(coluna, nomeFicheiro);
        strcat(coluna, " >> ficha03_mediana.dat");
        strcat(calculaMaximo, coluna);

        system(calculaMaximo);

        //calculaMinimo
        char calculaMinimo[100] = ("awk -F ' ' 'BEGIN{numeroMinimo=500}{if ($");
    }
}
```

Figura 22 - Função Calculos para o calculo da Mediana, Maximo e Minimo

Já os testes, terão os mesmos resultados obtidos na ficha 02, exceptuando as novas mensagens de sucesso de transmissão dos sinais, como por exemplo:

```
1,Processo de processo 2,Processo de processo 3,Signal
Sinal SIGUSR1 recebido pelo processo com PID 24185
```

Figura 23 - Mensagem de receção do sinal

## **5 Conclusão**

Damos assim por concluído a realização das 3 fichas de programação de SO, sendo que conseguimos fazer tudo e não temos críticas externas a fazer devido a um ótimo enunciado sem exigências desnecessárias.

### **5.1Strength**

Ao longo da realização das várias fichas foi sempre feito um estudo inicial daquilo que estávamos a abordar, o que facilitou bastante depois ao fazer os vários exercícios e ajudou a solidificar o que aprendemos ao longo das aulas.

### **5.2Weaknesses**

Como estamos a dar novos conceitos demorou algum tempo a adaptarmo-nos e causou algum atrito no início do trabalho.

### **5.3Opportunities**

Sem nada a indicar.

### **5.4Threats**

Sem nada a indicar.

## **6 Referências**

-[https://docs.oracle.com/cd/E37838\\_01/html/E60997/spprocess-30645.html#scrolltoc](https://docs.oracle.com/cd/E37838_01/html/E60997/spprocess-30645.html#scrolltoc)

-<https://www.geeksforgeeks.org/top-command-in-linux-with-examples/>

-<https://www.geeksforgeeks.org/signals-c-set-2/>

- PL - Programacao - 01 - Processes
- PL - Programacao - 02 - Processes - Parent-Child
- PL - Programacao - 03 - Signals