

(SO) Exercicios Prática

From WikiNote

Contents

- 1 Ficha 1
 - 1.1 a)
 - 1.2 b)
 - 1.3 c)
 - 1.4 d)
 - 1.5 f)
 - 1.6 g)
 - 1.7 h)
- 2 ifdef / endif
- 3 Compilar ficheiros
 - 3.1 Processo de compilação
 - 3.2 Como executar um programa em linux
 - 3.3 Divisão e eficácia no processo de compilação
- 4 GDB
 - 4.1 Fontes
- 5 Ficha 2
 - 5.1 Exercício 3
 - 5.1.1 a)
 - 5.1.2 b)
 - 5.1.3 c)
- 6 MAKE
 - 6.1 Exemplo de um make file
 - 6.2 Fontes
- 7 Args linha comandos
 - 7.1 Exemplo de programa
 - 7.2 Fontes
- 8 Variáveis de ambiente
 - 8.1 Exemplo de aplicação usando Variável de Ambiente
 - 8.2 Ficha 2
 - 8.2.1 Exercício 11
 - 8.2.1.1 a)
 - 8.2.1.2 b)
 - 8.2.1.3 d)
 - 8.3 Ficha 3
 - 8.3.1 Exercício 3
 - 8.3.1.1 a)
- 9 Processos
 - 9.1 Alguns comandos
 - 9.2 Ficha 3 EX4
 - 9.2.1 Código com o "execl" que estraga o nosso programa, pois não tem o fork()
 - 9.3 Ficha 3 EX4 com fork() e wait()
 - 9.4 Ficha 3 EX5
 - 9.5 Ficha 3 EX6
 - 9.5.1 ding
 - 9.5.2 dfran
 - 9.5.3 traduz
- 10 Sinais
 - 10.1 Exemplo de utilização do SIGINT (Ctrl + C)
 - 10.2 Exemplo de utilização do SIGALRM
 - 10.3 Exemplo de utilização do KILL
 - 10.4 Exercício 2 Ficha 4 Usando o signal
 - 10.5 Exercício 4 ficha 4 Usando o sigaction
 - 10.5.1 Nota sobre o siginfo
 - 10.6 Exercício 3 ficha 4
- 11 Redirecionamento
 - 11.1 Exemplo escrevendo num ficheiro
 - 11.2 Exemplo correto de redirecionamento
 - 11.3 Exemplo de redirecionamento com fork()
 - 11.4 mkfifo
 - 11.4.1 Ficha 5
 - 11.4.1.1 Exercício 3
- 12 Threads
 - 12.1 Notas
 - 12.2 Mutex (Trinco)
 - 12.3 Exercício 2 ficha 6
 - 12.3.1 Esquema
 - 12.3.2 envia.c
 - 12.3.3 ex2.c(servidor)

Ficha 1

a)

i)

```
man cp | man mv | man rm
```

ii)

```
man 3 printf -> função printf
```

b)**i)**

```
passwd
```

ii)

```
passwd
```

c)**i)**

```
ls
```

ii)

```
ls -la ou ls -l
```

iii)

```
ls -r
```

d)**i)**

```
cd /tmp
```

ii)

```
pwd
```

iii)

```
cd ..
```

iv)

```
ls /bin
```

v)

```
aulas  
cd aulas
```

vi)

```
mkdir aulas/a aulas/a/b
```

vii)

```
rm -r aulas/*  
    ou  
rm -r aulas/a/b aulas/a  
<pre/>  
  
== e) ==  
  
'''i)'''  
<pre>printenv PATH
```

ii)

```
export TESTE='aula123'
```

iii)

```
echo $TESTE
ou
printenv | grep TESTE
env | grep TEST
```

iv)

Uma vez que era uma variavel de ambiente, ao fazermos exit do terminal (ambiente) a variavel desaparece

f)**v)**

```
echo "ola"
```

vi) a)

nano ident e depois preencher o nome e numero

b)

```
echo -e "BrunoTeixeira\n2019100036" > ident (basicamente o -e assume que o \n é um breakline)
```

vii)

```
cat ident
```

g)**i)**

```
ls -l
```

ii)

Uma vez que o ficheiro não é executavel, usando o ls -l conseguimos ver que não existe o "x"

iii)

```
sudo chmod +x ident
```

iv)

```
./ident (ao usar o ./ estamos a executar um ficheiro)
```

v)

```
sudo chmod -rw ident
```

vii)

```
sudo chown man ident
```

viii)

```
sudo chown brun0 ident
```

ix)

```
cat /etc/passwd
```

x)

No conteudo /etc/shadow aparecem as credenciais encriptadas.
É preciso usar o sudo cat /etc/shadow uma vez que é um ficheiro protegido

xi)

```
su man
```

h)**i)**

```
cp ident dois
```

ii)

```
mv dois tres (o ficheiro dois passa a chamar-se tres)
```

iii)

```
rm tres
```

iv)

```
rm -r aula
```

v)

```
echo */ > lista
```

vi)

```
cat lista | wc -w
```

vii) a)

```
ls /bin/ | more
```

vii) b)

```
ls /bin/ | sort
```

vii) c)

```
ls /bin/ | grep "bin"
```

ifdef / endif

NOTA

#ifdef is a pre-processor directive that only compiles the code between *#ifdef* and *#endif* when the specified symbol (*DEBUG*) is defined by the compiler.

Compilar ficheiros

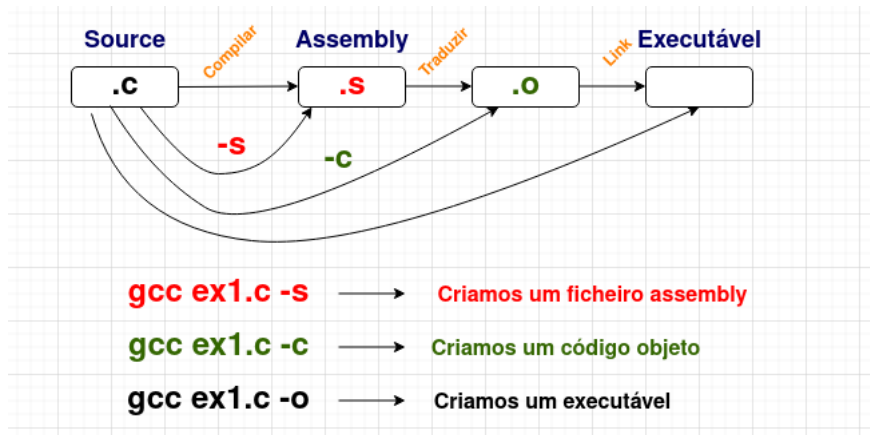
```
gcc ex1.c -o ex1
```

source exe

```
gcc -D
```

Serve para ver a flag DEBUG

Processo de compilação



Como executar um programa em linux

Para executar uma aplicação usamos o `./ex1` porque está na pasta corrente.

Se escrevermos no terminal só `ex1` o SO vai tentar encontrar o ficheiro das pasta do `$PATH`, logo não vai executar.

Por isso mesmo é que precisamos de dizer sempre que estamos a querer executar algo no directorio corrente usando `./ex1`.

Divisão e eficacia no processo de compilação

Podemos dividir o processo de código de fonte para o código executavel em vários passos.

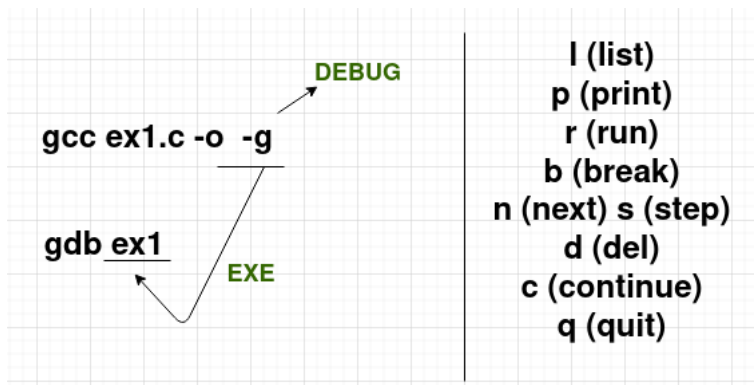
Quando temos vários ficheiros de codigo fonte (.c), é muito pouco provável que editemos os ficheiros todos em simultaneo, se não estamos a mexer nos outros ficheiros, não faz sentido fazer sempre o mesmo processo(compilar) para todos os ficheiros.

Logo o que podemos fazer é partir do código fonte gerar o código máquina (.o) para todos eles, no dia seguinte quando for trabalhar, se só for mexer com a interface do sistema ou editar alguma coisa simples só estamos a mexer num ficheiro de código e por isso não faz sentido estar a repetir o primeiro processo, faz mais sentido fazer o processo de codigo .c(código fonte) para código .o (código objeto) para o ficheiro que estamos a editar e na fase seguinte agarramos no programa que alteramos e geramos um objeto a seguir pego nos outros códigos objetos que não editei + o código objeto que editamos e gero o executável.

Usamos o **MAKE** para automatizar.

GDB

- O GDB (GNU Project Debugger) é uma ferramenta para:
 - observar um programa enquanto este executa
 - ver o estado no momento que a execução falha
- Permite
 - iniciar a execução de um programa
 - executar linha-a-linha
 - especificar pontos de paragem
 - imprimir valores de variáveis



Fontes

- <https://www.cs.umd.edu/~srhuang/teaching/cmsc212/gdb-tutorial-handout.pdf>

Ficha 2

Exercicio 3

a)

```
gcc programa.c imprime.c -o ex3
```

b)

```
gcc programa.c -c  
gcc imprime.c -c
```

c)

```
gcc programa.o imprime.o -o ex3
```

MAKE

O make serve para nos automatizar os processos de compilação.

- Antes de cada comando temos de colocar uma regra.
- Os comandos têm de ter um TAB antes.
- Antes de cada comando convem dizer em que condições o comando vai ser executado.
- Usa-se : como separador

Do lado esquerdo temos aquilo que queremos obter : Temos aquilo de que depende para obter o do lado esquerdo. Em baixo terá o comando de compilação para obter o lado esquerdo.

Como é que o make sabe que tem ou não de fazer o gcc?

Pelas datas. Se a hora/data dos programas da direita forem posterior ao da esquerda(.o) então ele sabe que tem de gerar um novo .o.

Quando não se coloca nada à frente da regra , essa regra será sempre executada como mostra em baixo no exemplo da regra **limpa**.

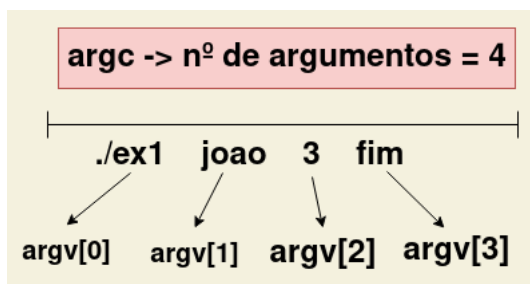
Exemplo de um make file

```
ex3:programa.o imprime.o  
    gcc programa.o imprime.o -o ex3  
  
programa.o:programa.c imprime.h  
    gcc programa.c -c  
  
imprime.o:imprime.c  
    gcc imprime.c -c  
  
limpa:  
    rm *.o
```

Fontes

- <https://makefiletutorial.com/>

Args linha comandos



O argv é um array em que cada elemento é uma string (ponteiro para char)

Exemplo de programa

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    int i;

    printf("[DEBUG] Inicio...\n");
    for (i=0; i<argc; i++) {
        printf("ARG[%d] = '%s'\n", i, argv[i]);
    }
    i = atoi(argv[2]);
    printf("Vou permitir mais 1... %d\n", ++i);
    printf("[DEBUG] Fim!\n");

    return(0);
}
```

Fontes

- https://www.gnu.org/software/libc/manual/html_node/Program-Arguments.html | Argumentos de linha de comandos

Variáveis de ambiente

Criamos uma variável de ambiente escrevendo no terminal **NOME="Joao Pedro"** e logo de seguida **export NOME**.

Temos de usar obrigatoriamente o **export** para ser considerada uma variável de ambiente.

Logo qualquer aplicação que executemos irá ter acesso à variável que acabamos de criar.

Exemplo de aplicação usando Variável de Ambiente

Para isto usamos o **char *envp[]** sendo um dos argumentos de uma função, neste caso vamos usar na main.

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[], char *envp[]) {
    int i;
    char *str;

    printf("[DEBUG] Inicio...\n");
    for (i=0; i<argc; i++) {
        printf("ARG[%d] = '%s'\n", i, argv[i]);
    }
    i = atoi(argv[2]);
    printf("Vou permitir mais 1... %d\n", ++i);
    for (i=0; envp[i]!=NULL; i++) {
        printf("VAR[%d] = '%s'\n", i, envp[i]);
    }
    str = getenv("NJOGADORES");
    i = atoi(str);
    printf("Vou permitir %d jogadores!\n", i);
    printf("[DEBUG] Fim!\n");

    return(0);
}
```

Se quisermos mostrar os argumentos todos

Converter uma string para um inteiro

Se quisermos mostrar as variáveis todas de ambiente

Se quisermos ir buscar uma variável de amb em específico

Ficha 2

Exercício 11

The printenv and env commands print only the environment variables. If you want to get a list of all variables, including environment, shell and variables, and shell functions you can use the set command.

a)

set

b)

```
int main(int argc, char const *argv[], char *envp[])
{
    int i;
    char *str;

    for(i = 0; envp[i] != NULL; i++){
        printf("%s\n", envp[i]);
    }

    return 0;
}
```

A screenshot of a terminal window titled 'brun0@debian: ~/codigo_aula'. The terminal displays a list of environment variables in green text: TERM=xterm-256color, USER=brun0, SHLV=1, XDG_SESSION_ID=2, XDG_RUNTIME_DIR=/run/user/1000, SSH_CLIENT=192.168.1.123 54314 22, PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games, DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus, MAIL=/var/mail/brun0, SSH_TTY=/dev/pts/0, _=./ex11e, OLDPWD=/home/brun0. The prompt 'brun0@debian:~/codigo_aula\$' is visible at the bottom.

d)

```
int main(int argc, char *argv[], char *envp[]){
    int i;
    char *str;

    str = getenv("NOME");
    printf("O meu nome é %s\n", str);

    return 0;
}
```

/*no .bashrc está criada uma variavel de ambiente

```
export NOME
NOME="Bruno Teixeira"
*/
```

A screenshot of a terminal window titled 'brun0@debian: ~/codigo_aula/27_10_2020'. The terminal shows the command './ex11' being executed, which outputs 'O meu nome é Bruno Teixeira' in green text. The prompt 'brun0@debian:~/codigo_aula/27_10_2020\$' is visible at the bottom.

Ficha 3

Exercicio 3

a)


```

int main(int argc, char const *argv[])
{
    int i,n,t = 1;
    char *str;

    printf("INICIO...\n");

    if(argc != 3){
        printf("[ERRO] Nr de args inválido !\n");
        exit(3);
    }

    n = atoi(argv[1]);
    str = getenv("TEMPO");

    if(str != NULL){
        t = atoi(str);
    }

    for(i = 0; i < n ; i++){
        printf("%s", argv[2]);
        fflush(stdout);
        sleep(t);
    }

    printf("\nFIM...");

    exit(0);
}
/* o exit() diz que eu quero parar com a aplicação
estou a terminar aplicação onde quer que esteja*/

/*o sleep() permite fazer uma pausa na aplicação
durante N segundos*/

/* o fflush() permite forçar a saída do que está
pendente do que tem de sair para o ecrã */

```

```

brun0@debian: ~/codigo_aula/27_10_2020
File Edit View Search Terminal Help
brun0@debian:~/codigo_aula/27_10_2020$ ./ex3estudo
INICIO...
*****
FIM...brun0@debian:~/codigo_aula/27_10_2020$

```

Processos

```

brun0@debian:~/codigo_aula/27_10_2020$ ps ax
  PID TTY          STAT       TIME COMMAND
  1 ?        Ss         0:01 /sbin/init
  2 ?        S           0:00 [kthreadd]

```

O PID 1 e 2 são os dois primeiros processos do sistema.
Ambos não têm pai.
Todos os outros processos normalmente filhos destes dois processos

Todos os processos que têm parênteses retos são filhos do
[kthreadd]
Normalmente todos os processos em modo de utilizador
são filhos do /sbin/init

Quando um pai de um processo morre, normalmente esse processo
é adotado pelo /sbin/init

- Tarefa -> terminal
- Processo -> sistema

O "ps" retorna tudo o que está a correr um bloco de código, no entanto os "jobs" mostram todas as tarefas, por exemplo escrever no terminal.

O "jobs" mostra todas as tarefas e os estados delas.

O "ps a" serve para ver que processos é que estou a correr em meu nome

```
ps a
```

Para ver quem é o pai do processo basta ver o "PPID".

Podemos usar o "ps f" e mostra logo quem é o pai e o filho.

Alguns comandos

- Se executarmos o ex3 da ficha 3 por exemplo e fizermos ^Z , a aplicação para. Se fizermos "fg" a aplicação volta ao ativo do ponto onde tinha parado com o ^Z.
- Se fizermos "bg %1 , bg %2 ... retomamos todas as tarefas que estavam em "fg"
- Para terminarmos uma tarefa, podemos fazer "kill %id_da_tarefa"
- Podemos terminar uma aplicação também através do PID , basta fazermos "kill -9 PID_da_aplicação"

ps -> process status

jobs -> "tarefas"

fg -> foreground

bg -> background

kill -> "terminar"

^C -> "terminar"(SIGINT)

^Z -> "suspende"(SIGSTOP)

& -> lança de imediato a aplicação em bg

Ficha 3 EX4

Quanto fazemos um "execl" no código, estragamos o nosso 1º código. Para não estragar, temos de criar um filho e quem irá fazer o "execl" é o filho.

Código com o "execl" que estraga o nosso programa, pois não tem o fork()

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char const *argv[])
{
    char str[40];

    do{
        printf("COMANDO:");
        fflush(stdout);
        scanf("%s", str);
        execl(str, str, NULL);
        printf("[ERR0] Nao consegui executar aplicação\n");
    }while(strcmp(str,"sair") != 0);

    exit(0);
}

/*Depois do primeiro parametro no execl
damos todos os argumentos que queremos,
terminados em NULL*/

/*se o execl for bem sucedido, este código
irá desaparecer e mesmo com o do while
a shell nao volta a pedir o COMANDO*/

/*se o execl falha, ocorre o [ERR0]*/
```

exec("path do executavel", "ex3", "15", "+", NULL);

O processo do ex4 passa a ser o processo do ex3, logo o ex4 desaparece caso o execl funcione.

getpid(); -> dá o meu PID

getppid(); -> dá o Parent PID (Pai)

exec(); -> executa um executável e perde o código original

Ficha 3 EX4 com fork() e wait()

```
int main(int argc, char const *argv[])
{
    char str[40];
    int continua = 1 ,pid,res, estado;

    pid = getpid(); // mostra o PID do processo atual

    while(continua){
        printf("[%5d] COMANDO:",pid);
        fflush(stdout);
        scanf("%s", str);
        if(strcmp(str, "sair") != 0){
            res = fork(); //cria um processo filho
            if(res == 0){ // se for verdade, sabemos que isto é o filho
                pid = getpid(); // recebemos o PID do filho
                printf("[%5d] FILHO: Sou o filho ...\n",pid);
                execl(str, str, NULL); // se o execl for bem sucedido, o proximo printf não aparece
                printf("[%5d] [ERRO] FILHO: Nao consegui executar aplicação\n",pid);
                exit(7);//o exit aqui serve para se algo correr mal
            }
            printf("[%5d] PAI : Criei o filho PID=%d\n",pid,res);
            wait(&estado); //esperar pelo fim do filho e devolve na var estado a forma como o filho acabou
            if(WIFEXITED(estado)){
                printf("[%5d] PAI : O meu filho terminou com %d\n",pid,WEXITSTATUS(estado));
            }
        }
        else{
            continua = 0;
        }
    }

    exit(0);
}

/*no fork() se o resultado é 0 é filho, se o resultado for != 0 é pai*/

/*para saber que filho é que morreu usamos int n = wait(&estado)
o "n" vai retornar o pid do filho que morreu*/

/*convem usar sempre wait() para o pai esperar pelo filho*/

/* WIFEXITED -> Retorna true se o filho terminar normalmente
WEXITSTATUS -> retorna o exit status do filho */
```

Ficha 3 EX5

```
int main(int argc, char const *argv[])
{
    int a = 10;

    if (fork() == 0)
        a++;
    else
        a--;
    printf("\na = %d\n", a);

    return 0;
}

/*a = 9 e a = 11 , a ordem é desconhecida
porque o SO não me garante nada*/
```

Ficha 3 EX6

ding

```

int main(int argc, char const *argv[])
{
    int i;
    char str[40];

    char *cadeiap[1][5] = {
        "gato", "comida", "peixe", "cao", "animal"
    };

    char *cadeiai[1][5] = {
        "cat", "food", "fish", "dog", "pet"
    };

    printf("[INICIO]\n");

    strcpy(str,argv[1]);

    for(i = 0; i < 5; i++){
        if(strcmp(str,cadeiap[0][i]) == 0){
            printf("%s\n", cadeiai[0][i]);
            printf("[FIM]\n");
            exit(1);
        }
    }

    printf("Unkown\n");

    printf("[FIM]\n");

    exit(0);
}

```

dfran

```

int main(int argc, char const *argv[])
{
    int i;
    char str[40];

    char *cadeiap[1][5] = {
        "gato", "comida", "peixe", "cao", "animal"
    };

    char *cadeiaf[1][5] = {
        "chat", "nourriture", "poisson", "chien", "animal"
    };

    printf("[INICIO]\n");

    strcpy(str,argv[1]);

    for(i = 0; i < 5; i++){
        if(strcmp(str,cadeiap[0][i]) == 0){
            printf("%s\n", cadeiaf[0][i]);
            printf("[FIM]\n");
            exit(1);
        }
    }

    printf("Unkown\n");

    printf("[FIM]\n");

    exit(0);
}

```

traduz

```

int main(int argc, char const *argv[])
{
    int continua = 1, res, estado;
    char letra;
    char palavra[10];

    do{
        printf("Letra: "); //pede uma letra
        fflush(stdout);
        scanf(" %c", &letra);
        if(letra == 'i'){ //se for i, vai ao "ding"
            printf("Palavra: ");
            fflush(stdout);
            scanf("%s", palavra); // recebe a palavra que ira ser passada como argumento para o "ding"
            res = fork(); //cria o filho
            if(res == 0){
                execl("ding", "ding", palavra, NULL); // se o execl correr bem, executa o ding
                printf("erro\n"); // se correr mal, dá um erro e faz-se exit
                exit(7);
            }
            wait(&estado); // o pai espera pelo filho
            if(WIFEXITED(estado)){ //e vê como é que ele terminou
                printf("O filho terminou com %d\n", WEXITSTATUS(estado));
                fflush(stdout);
            }
        }
        else if(letra == 'c'){ // se a letra for a c, vai ao "dfran"
            printf("Palavra: ");
            fflush(stdout);
            scanf("%s", palavra);
            res = fork(); //cria o filho
            if(res == 0){
                execl("dfran", "dfran", palavra, NULL); // se o execl correr bem, executa o dfran
                printf("Erro no fork()\n"); // se nao, aparece um erro
                exit(7); // e fazemos exit(7)
            }
            wait(&estado); // o pai espera pelo filho
            if(WIFEXITED(estado)){ // e vê como é que ele terminou
                printf("O filho terminou com %d\n", WEXITSTATUS(estado));
                fflush(stdout);
            }
        }
    }while(letra != 'x');

    printf("FIM...\n");

    return 0;
}

```

Sinais

signal() → Informa ao Sistema Operativo o que é que eu quero fazer quando chega o sinal

kill() → Serve para enviar um sinal a um processo

pause() → Serve para dormir até à chegada de um sinal

alarm() → Serve para programar a entrada de um sinal (sigalrm)

Pode-se definir 3 comportamentos para sinais:

- Terminar a aplicação
- Ignorar o sinal
- Associar a uma função

O SIGKILL e o SIGSTOP não dão para ignorar ou associar a uma função

Exemplo de utilização do SIGINT (Ctrl + C)

```
// nao se deve fazer| texto dentro desta funcao
// porque é a funcao de tratamento de sinal e tem de ser rapida
void mostra(int s){
    printf("\nOla!!! (recebi o sinal %d)\n",s);
}

int main(int argc, char const *argv[])
{
    char str[40];

    //signal(SIGINT, SIG_IGN) // ignora o sinal SIGINT
    signal(SIGINT, mostra); // associa o sinal a uma função
    printf("O meu PID e %d ...\n",getpid());

    do{
        printf("Nome: ");
        fflush(stdout);
        scanf("%s", str);
        printf("Ola");
        for(int i = 0; i < 2 ; i++){
            printf("a");
            fflush(stdout);
            sleep(1);
        }
        printf(" %s!!!\n",str);
    }while(strcmp(str, "sair") != 0);

    exit(0);
}
```

Exemplo de utilização do SIGALRM

```
void mostra(int s){
    printf("\nAcorda!!! (sinal %d)",s);
    fflush(stdout);
}

int main(int argc, char const *argv[])
{
    char str[40];

    signal(SIGALRM, mostra); // Neste caso o quando for chamado o alarme, a funcao "mostra" vai executar
    printf("O meu PID e %d ...\n",getpid());

    do{
        alarm(10); // alarme de 10 segundos
        printf("Nome: ");
        fflush(stdout);
        scanf("%s", str);
        alarm(0); // quando o utilizador responder desligamos o alarme
        printf("Ola");
        for(int i = 0; i < 2 ; i++){
            printf("a");
            fflush(stdout);
            sleep(1);
        }
        printf(" %s!!!\n",str);
    }while(strcmp(str, "sair") != 0);

    exit(0);
}

/*com o alarm(10), estou a dizer ao sistema operativo assim
"daqui a 10 segundos da-me um toque" sendo este toque o
SIGALRM

Por omissão o SIGALRM depois de ser chamado vai terminar
*/
```

Exemplo de utilização do KILL

```


int main(int argc, char const *argv[])
{
    int pid,sinal;

    if(argc != 3){
        fprintf(stderr, "[ERR0] Nr de argumentos !\n          ./envia SINAL PID\n");
        exit(1);
    }

    sinal = atoi(argv[1]);
    pid = atoi(argv[2]);

    printf("Vou enviar o sinal %d ao processo %d...\n", sinal,pid);
    kill(pid,sinal); // envia um sinal ao processo
    exit(0);
}

```



Exercicio 2 Ficha 4 Usando o signal

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <signal.h>
#include <time.h>

void mostra(int s){
    printf("\nACORDA!!! (sinal %d)\n",s);
    alarm(10);
}

int main(int argc, char const *argv[])
{
    int i,res, nerradas = 0, ncertas = 0, tempo = 10,num1,num2;
    char str[40];

    srand((unsigned int) time(NULL));
    signal(SIGALRM, mostra);
    printf("O meu PID e %d...\n",getpid());

    do{
        num1 = rand() % 101;
        num2 = rand() % 101;
        printf("Tem %d segundos...\n",tempo);
        printf("%d + %d ?\n", num1, num2);
        fflush(stdout);
        alarm(tempo);
        scanf("%d", &res);
        alarm(0);

        if(res == num1+num2){
            ncertas++;
            tempo--;
        }
        else{
            nerradas++;
        }
    }while(strcmp(str,"sair")!= 0);

    exit(0);
}

```



Ao usar o signal ficamos parados no scanf

Exercicio 4 ficha 4 Usando o sigaction

```
void mostra(int s){
    printf("\nACORDA!!! (sinal %d)\n",s);
    nerradas++;
}

int main(int argc, char const *argv[])
{
    int i,res, nerradas = 0, ncertas = 0, tempo = 10,num1,num2;
    char str[40];

    struct sigaction act;
    act.sa_handler = mostra; // funcao a executar //
    act.sa_flags = 0; // nao faz o restart//

    sigaction(SIGALRM,&act,NULL);

    srand((unsigned int) time(NULL));
    printf("0 meu PID e %d...\n",getpid());

    do{
        num1 = rand() % 101;
        num2 = rand() % 101;
        printf("Tem %d segundos...\n",tempo);
        printf("%d + %d ?\n", num1, num2);
        fflush(stdout);
        alarm(tempo);
        scanf("%d", &res);
        alarm(0); // Quando chegar aqui, volta ao inicio do ciclo

        if(res == num1+num2){
            ncertas++;
            tempo--;
        }
        else{
            nerradas++;
        }

    }while(strcmp(str,"sair")!= 0);

    exit(0);
}
```

Nota sobre o siginfo


```

void nova(int s, siginfo_t *info, void *context){
    printf("\nACORDA!!! (Sinal = %d, PID = %d, Valor = %d)\n", s, info->si_pid, info->si_value);

    if(info->si_pid == 0) /* se alguém fora da minha aplicação enviar-me sinais, eu não
                           quero contar erradas*/
        nerradas++;
}

int main(int argc, char const *argv[])
{
    int i, res, nerradas = 0, ncertas = 0, tempo = 10, num1, num2;
    char str[40];

    struct sigaction act;
    act.sa_sigaction = nova; // função a executar
    act.sa_flags = SA_SIGINFO; // queremos receber informação adicional

    sigaction(SIGALRM, &act, NULL);

    srand((unsigned int) time(NULL));
    printf("O meu PID é %d...\n", getpid());

    do{
        num1 = rand() % 101;
        num2 = rand() % 101;
        printf("Tem %d segundos...\n", tempo);
        printf("%d + %d ?\n", num1, num2);
        fflush(stdout);
        alarm(tempo);
        scanf("%d", &res);
        alarm(0); // Quando chegar aqui, volta ao início do ciclo

        if(res == num1+num2){
            ncertas++;
            tempo--;
        }
        else{
            nerradas++;
        }
    }while(strcmp(str, "sair") != 0);

    exit(0);
}

```

Exercício 3 ficha 4

```

int main(int argc, char const *argv[])
{
    int pid, sinal;

    if(argc != 3){
        fprintf(stderr, "[ERRO] Nr de argumentos!\n      ./envia SINAL PID");
        exit(1);
    }

    sinal = atoi(argv[1]);
    pid = atoi(argv[2]);

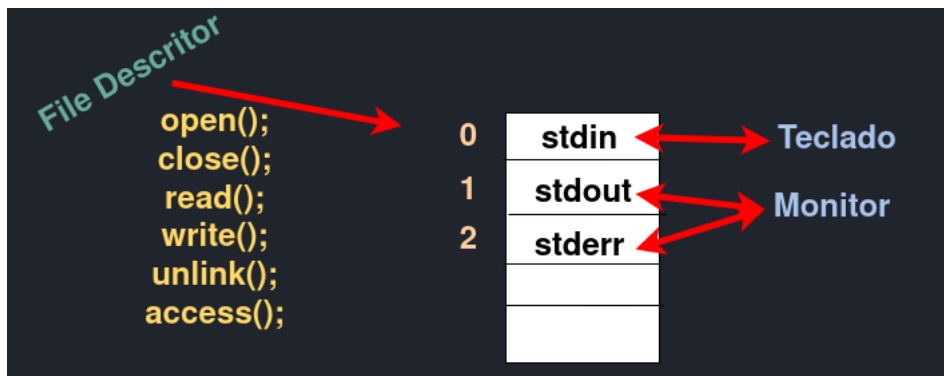
    printf("Vou enviar o sinal %d ao processo %d...\n", sinal, pid);

    union sigval value;
    value.sival_int = 666;
    sigqueue(pid, sinal, value); // envia sinal ao processo com um valor

    exit(0);
}

```

Redirecionamento



```

brun0@debian:~$ ls -l /proc/1321/fd
total 0
lrwx----- 1 brun0 brun0 64 Nov 23 23:20 0 -> /dev/pts/0
lrwx----- 1 brun0 brun0 64 Nov 23 23:20 1 -> /dev/pts/0
lrwx----- 1 brun0 brun0 64 Nov 23 23:20 2 -> /dev/pts/0
  
```

PID do processo

stdin
stdout
stderr

Exemplo escrevendo num ficheiro

```

int main(void){
    int i,fd;

    /*0_CREAT -> cria o ficheiro caso nao exista
    0_TRUNC -> limpa o ficheiro caso exista algum conteudo
    0_WRONLY -> para escrever no ficheiro
    0644 -> rw-r--r--

    */
    fd = open("dados.bin", O_CREAT | O_TRUNC | O_WRONLY, 0655);
    if(fd >= 0){

        printf("O meu PID e %d.\n",getpid());
        printf("Inicio...\n");
        for(i = 0; i < 10; i++){
            printf("+");
            fflush(stdout);
            sleep(2);
        }
        printf("\nFim!!\n");

        write(fd,"Bruno Teixeira",14);
        write(fd, &i,sizeof(int));
        close(fd);
    }

    exit(3);
}
  
```

```

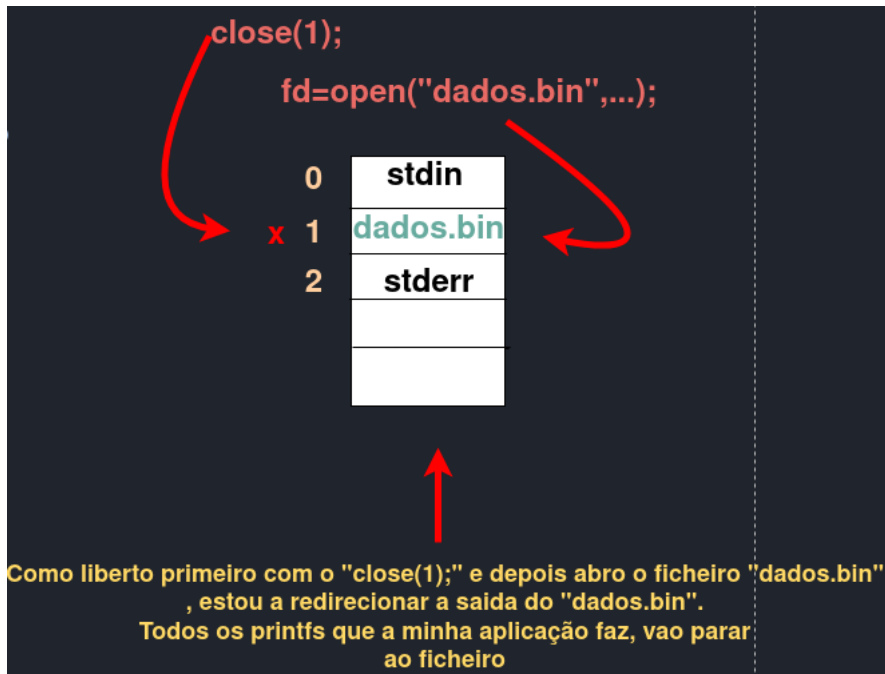
lrwx----- 1 brun0 brun0 64 Nov 23 23:48 0 -> /dev/pts/0
lrwx----- 1 brun0 brun0 64 Nov 23 23:48 1 -> /dev/pts/0
lrwx----- 1 brun0 brun0 64 Nov 23 23:48 2 -> /dev/pts/0
lrwx----- 1 brun0 brun0 64 Nov 23 23:48 3 -> /home/brun0/codigo_aula/17_11_2020_estudo/dados.bin
  
```

```

brun0@debian:~/codigo_aula/17_11_2020_estudo$ hexdump -C dados.bin
00000000  42 72 75 6e 6f 20 54 65  69 78 65 69 72 61 0a 00  |Bruno Teixeira..|
00000010  00 00                                     |..|
00000012
  
```

Exemplo correto de redirecionamento

O objetivo é redirecionar os printf's do programa para o ficheiro



```
int main(void){
    int i,fd;

    /*0_CREAT -> cria o ficheiro caso nao exista
    0_TRUNC -> limpa o ficheiro caso exista algum conteudo
    0_WRONLY -> para escrever no ficheiro
    0644 -> rw-r--r--

    */

    printf("0 meu PID e %d.\n",getpid());
    close(1); Fecho o stdout

    fd = open("dados.bin", O_CREAT | O_TRUNC | O_WRONLY, 0655);
    if(fd >= 0){

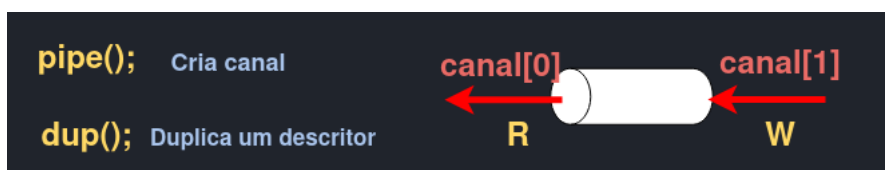
        printf("Inicio...\n");
        for(i = 0; i < 10; i++){
            printf("+");
            fflush(stdout);
            sleep(2);
        }
        printf("\nFim!!\n");

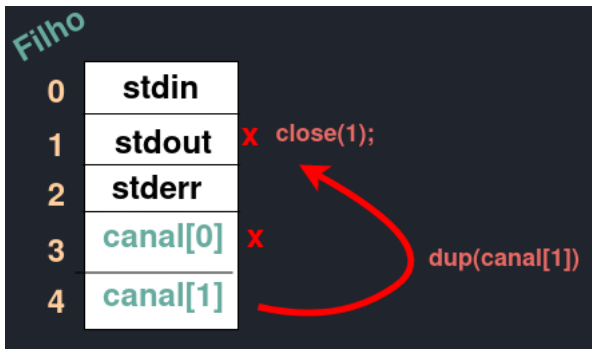
        write(fd,"Bruno Teixeira",14);
        write(fd, &i,sizeof(int));
        close(fd);
    }

    exit(3);
}
```

```
brun0@debian:~$ ls -l /proc/1410/fd
total 0
lrwx----- 1 brun0 brun0 64 Nov 24 00:05 0 -> /dev/pts/0
lrwx----- 1 brun0 brun0 64 Nov 24 00:05 1 -> /home/brun0/codigo_aula/17_11_2020_estudo/dados.bin
lrwx----- 1 brun0 brun0 64 Nov 24 00:05 2 -> /dev/pts/0
```

Exemplo de redirecionamento com fork()





cria o canal -> int canal[2]

prepara o canal -> pipe(canal)

cria o filho -> fork()

dentro do filho:

engana o jogo (redirecionamento)

fecha o canal[0] do filho porque o filho só quer escrever -> close(canal[0])

fecha o stdout porque queremos enviar a informação pelo pipe -> close(1)

duplica o canal para que o canal[1] (write do filho) apareça no stdout -> dup(canal[1])

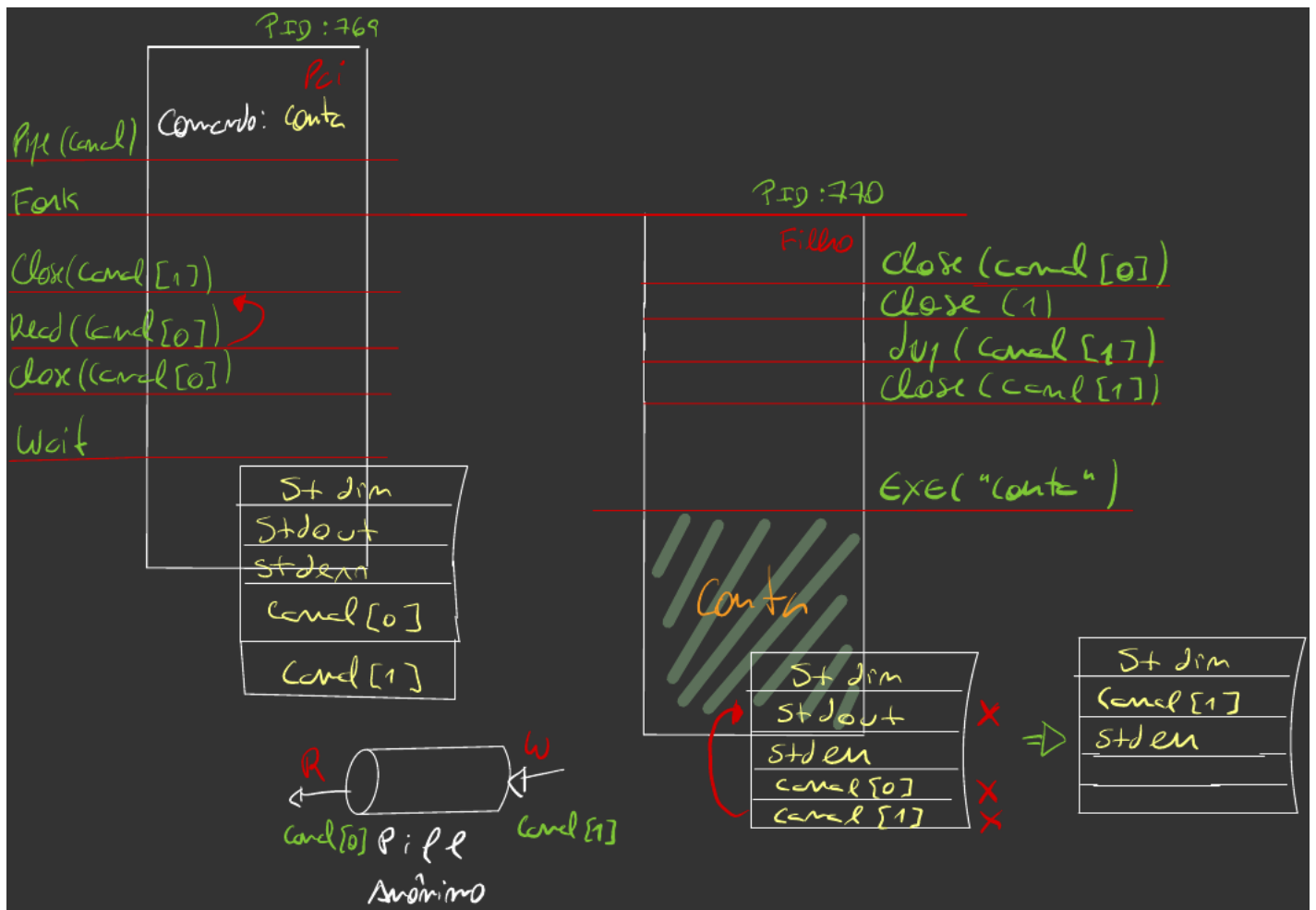
fecha o canal[1] uma vez que já foi duplicado -> close(canal[1])

fora do filho:

no pai fechamos o canal 1 porque não queremos escrever no pipe -> close(canal[1])

fazemos um while enquanto o canal[0] está a receber informação, assim o pai está à espera de informação

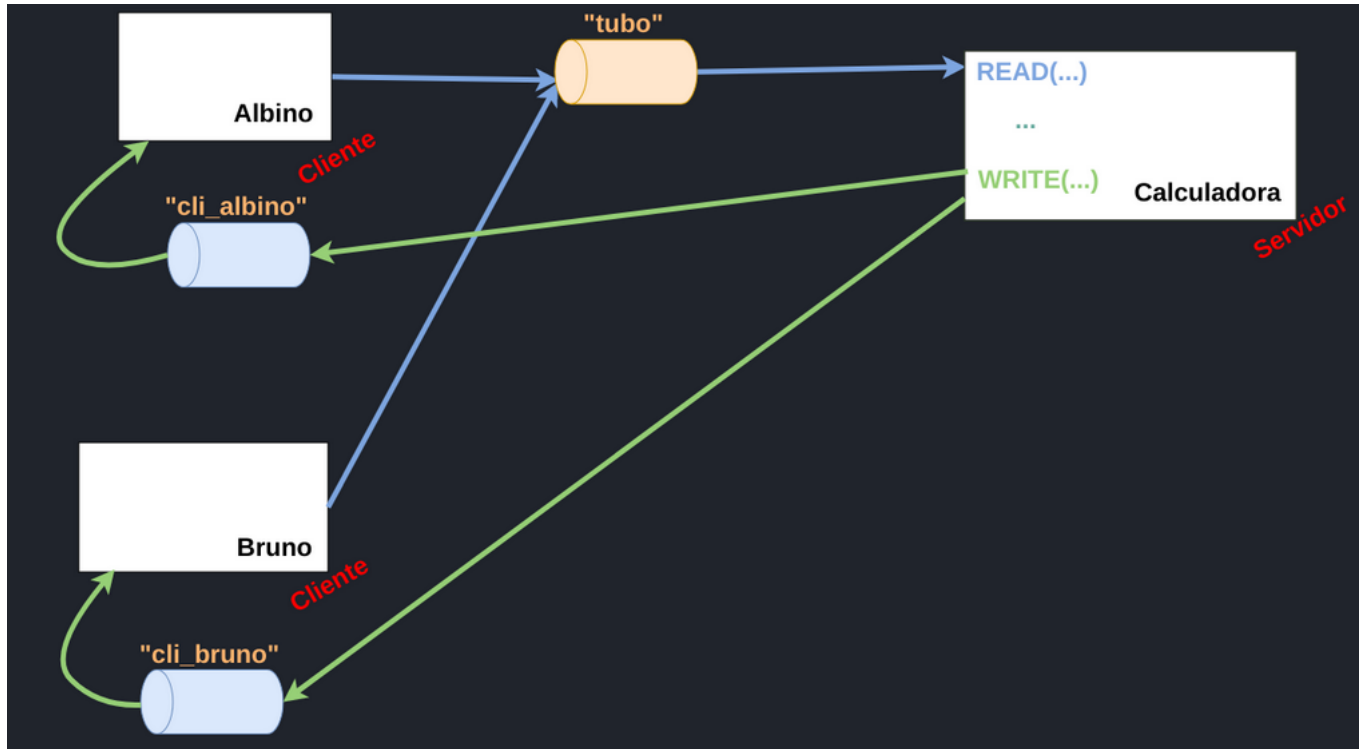
quando acabar de receber informação fechamos o canal 0 -> close(canal[0])



mkfifo

Ficha 5

Exercício 3



Threads

Notas

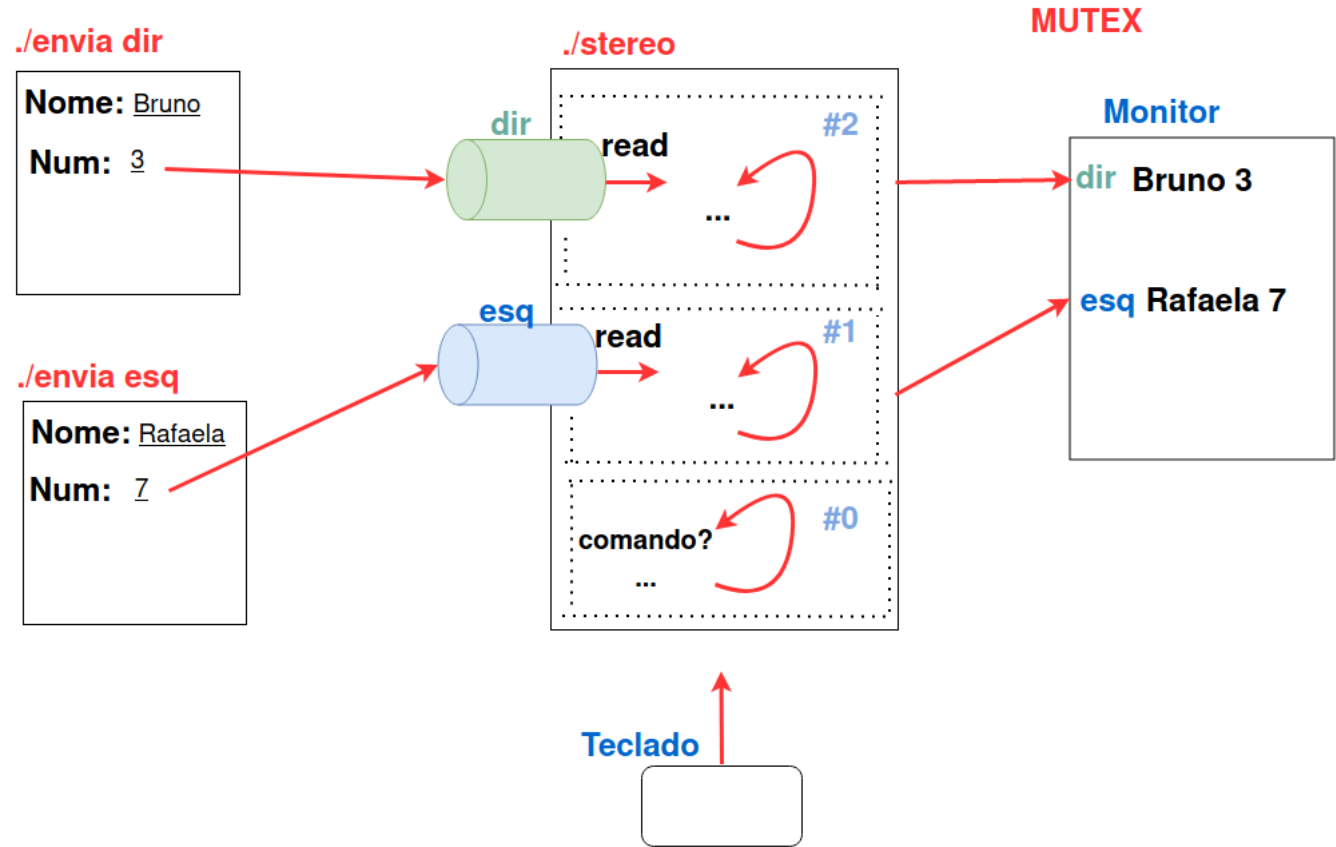
- Para compilar um programa com threads colocamos o gcc da seguinte maneira **gcc ex1.c -o ex1 -lpthread**
- Quando um processo é lançado é criada uma thread
- Quando se faz **exit()** numa thread, estamos a matar o processo todo e não é isso que queremos
- Os dados num programa são partilhados entre threads por isso é preciso ter cuidado com isso
- Se fizermos **ps alm** conseguimos ver as threads
- Quando temos várias threads ao mesmo tempo existe uma concorrência para acessos aos recursos
- Para sincronizar os processos usamos os Mutexes

Mutex (Trinco)

- Não esquecer que ao fazermos **pthread_mutex_lock()** temos de fazer depois **pthread_mutex_unlock()** obrigatoriamente
- Se a thread precisar de um trinco (mutex), podemos colocar um ponteiro por exemplo **pthread_mutex_t *ptrinco** na estrutura da thread

Exercício 2 ficha 6

Esquema



envia.c

```
#include "util.h"

int continua = 1;

/*funcao de tratamento de sinal*/
void termina(int s){
    continua = 0;
}

int main(int argc, char const *argv[]){
    PEDIDO p;
    int fd_serv;
    setbuf(stdout, NULL);

    struct sigaction act;
    act.sa_handler = termina;
    act.sa_flags = 0;

    /*VERIFICA SE TEM ARGUMENTOS DA LINHA DE COMANDOS*/
    if(argc != 2)
        exit(1);

    /*ABRE FIFO DO ARGV[1] PARA ESCRITA*/
    fd_serv = open(argv[1], O_WRONLY);

    if(fd_serv == -1)
        exit(1);

    /*PEDE NOME*/
    printf("Nome: "); scanf("%s", p.nome);
    /*GUARDA PID*/
    p.pid = getpid();
    /*TRATA DO SINAL*/
    sigaction(SIGUSR1, &act, NULL);

    do{
        /*PEDE NUMERO*/
        printf("Numero: "); scanf("%d", &p.num);

        /*ESCREVE NO FIFO*/
        write(fd_serv, &p, sizeof(PEDIDO));

    }while(continua);
    /*FECHA O FIFO*/
    close(fd_serv);
    exit(0);
}
```

ex2.c(servidor)

```

#include "util.h"

typedef struct {
    char fifo[20];
    int continua;
    pthread_mutex_t *ptrinco;
}TDATA;

void acorda(int s){}

void *mostra(void *dados){
    int *res,fd, n;
    struct sigaction act;
    PEDIDO p;

    TDATA *pdata;
    pdata = (TDATA *)dados; // vamos usar o pdata como argumento desta funcao
    res = (int *) malloc(sizeof(int)); // nr de pedidos chegados do cliente
    *res = 0;

    /*TRATAR O SINAL SIGUSR1 - sigaction*/
    act.sa_handler = acorda;
    act.sa_flags = 0;

    sigaction(SIGUSR1,&act,NULL);

    /*CRIAR FIFO - mkfifo()*/
    mkfifo(pdata->fifo,0600);

    /*ABRIR FIFO (O_RDWR) - open()*/
    fd = open(pdata->fifo,O_RDWR);

    do{
        /*RECEBER INFORMACAO DO CLIENTE - read()*/
        n = read(fd,&p, sizeof(p));
        if(n == sizeof(PEDIDO)){
            /*FAZEMOS LOCK NO MUTEX*/
            pthread_mutex_lock(pdata->ptrinco);

            /*ENVIAR INFORMACAO PARA O MONITOR - printf()*/
            printf("%s - %s %d\n",pdata->fifo,p.nome,p.num);

            /*FAZEMOS UNLOCK COM O MUTEX*/
            pthread_mutex_unlock(pdata->ptrinco);

            (*res)++;
        }
    }while(pdata->continua);
}

```



```

/*AVISAR O CLIENTE PARA SAIR - kill()*/
kill(p.pid,SIGUSR1);

/*FECHAR FIFO - close()*/
close(fd);

/*ELIMINAR FIFO - unlink()*/
unlink(pdata->fifo);

pthread_exit(res);
}

int main(void){
    char str[40];
    int *resultado;
    TDATA tinfo[2];
    pthread_t tarefa[2];
    pthread_mutex_t trinco;

    /*INICIALIZA O MUTEX*/
    pthread_mutex_init(&trinco,NULL);

    printf("Sou o processo %d...\n",getpid());

    /*ENVIAR UMA STRING A THREAD - "ESQ" ou "DIR"*/
    strcpy(tinfo[0].fifo,"ESQ");
    tinfo[0].continua = 1;
    tinfo[0].ptrinco = &trinco;
    pthread_create(&tarefa[0],NULL,mostra,(void *)&tinfo[0]);

    /*ENVIAR UMA STRING A THREAD - "ESQ" ou "DIR"*/
    strcpy(tinfo[1].fifo,"DIR");
    tinfo[1].continua = 1;
    tinfo[1].ptrinco = &trinco;
    pthread_create(&tarefa[1],NULL,mostra,(void *)&tinfo[1]);

    do{
        /*tarefa #0*/
        printf("COMANDO:\n");
        scanf("%s", str);
        printf("Recebi o comando '%s'...\n",str);

    }while(strcmp(str,"sair")!= 0);

```

```

/*AQUI ESTOU A TERMINAR AS DUAS THREADS*/
for(int i = 0; i < 2; i++){
    tinfo[i].continua = 0;

    /*ENVIAR SINAL SIGUSR1 A THREAD PARA DESBLOQUEAR O READ - pthread_kill()*/
    pthread_kill(tarefa[i],SIGUSR1);

    pthread_join(tarefa[i],(void *)&resultado); // esperar pelo fim da thread que criei
    printf("Thread %d terminou [%d]!!!\n", i,*resultado);

    free(resultado);
}

/*APAGAR O MUTEX*/
pthread_mutex_destroy(&trinco);

exit(0);
}

```

Retrieved from "[http://zebisnaga.pt/wiki/index.php?title=\(SO\)_Exercicios_Prática&oldid=1217](http://zebisnaga.pt/wiki/index.php?title=(SO)_Exercicios_Prática&oldid=1217)"