Daniel Schreiber Guimarães - 1910462 - INF1316 - Sinais

Questão 1:

```
pi@raspberrypi:/mnt/rede/Daniel/rasc/sinais $ sudo gcc -o teste ctrlc.c
pi@raspberrypi:/mnt/rede/Daniel/rasc/sinais $ ./teste
Endereco do manipulador anterior (nil)
Endereco do manipulador anterior (nil)
CTRL-C desabilitado. Use CTRL-\ para terminar
^CPressionou CTRL-C (2)
^CPressionou CTRL-C (2)
^CPressionou CTRL-C (2)
^CPressionou CTRL-C (2)
^CPressionou CTRL-C (2)
^CPressionou CTRL-C (2)
^\Terminando o processo!
```

Ao tentar digitar CTRL+C, o processo respondia de acordo. Ao digitar CTRL+\, o processo fechou.

```
pi@raspberrypi:/mnt/rede/Daniel/rasc/sinais $ ./teste
Endereco do manipulador anterior (nil)
Endereco do manipulador anterior (nil)
CTRL-C desabilitado. Use CTRL-\ para terminar
^C
pi@raspberrypi:/mnt/rede/Daniel/rasc/sinais $
```

Ao retirar o `p = signal(....);` o processo não capturou o sinal, e foi terminado.

Questão 2:

Código da tentativa:

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

#define EVER ;;

void killHandler(int sinal);

int main()
{
    void (*p)(int);
    p = signal(SIGKILL, killHandler);
    printf("Endereco do manipulador anterior %p\n", p);
    printf("Tente matar o processo!\n");
    for(EVER);
}

void killHandler(int sinal) {printf("Impossivel de matar!");}
```

Rodando o programa “imortal”, verifiquei qual era o PID dele, e enviei um sinal de *kill* para o processo, e ele foi morto, em vez de exibir “Impossível de matar!”. Ou seja, não foi possível evitar a morte. Isso acontece pois o sinal de *kill* é impossível de ser ignorado.

```
pi@raspberrypi:/mnt/rede/Daniel/rasc/sinais $ ./imortal &
[1] 16840
pi@raspberrypi:/mnt/rede/Daniel/rasc/sinais $ Endereco do manipulador anterior 0xffffffff
Tente matar o processo!
ps
  PID TTY          TIME CMD
16538 pts/0    00:00:00 bash
16840 pts/0    00:00:19 imortal
16841 pts/0    00:00:00 ps
pi@raspberrypi:/mnt/rede/Daniel/rasc/sinais $ kill -s SIGKILL 16840
pi@raspberrypi:/mnt/rede/Daniel/rasc/sinais $ ps
  PID TTY          TIME CMD
16538 pts/0    00:00:00 bash
16846 pts/0    00:00:00 ps
[1]+  Killed                  ./imortal
pi@raspberrypi:/mnt/rede/Daniel/rasc/sinais $
```

Questão 3:

Aqui, estou fazendo o filho executar para sempre. Ao executar com um tempo de 2 segundos, o processo pai mata o filho, pois demorou mais que 2 segundos.

```
pi@raspberrypi:/mnt/rede/Daniel/rasc/sinais $ ./filhocidio 2
programa excedeu o limite de 2 segundos
pi@raspberrypi:/mnt/rede/Daniel/rasc/sinais $
```

Como funciona o código:

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

    signal(SIGCHLD, childhandler);
    if ((pid = fork()) < 0)
    {
        fprintf(stderr, "erro ao criar filho\n");
        exit(-1);
    }
    if (pid == 0) {for(EVER);}
    else
    {
        sscanf(argv[1], "%d", &delay);
        sleep(delay);
        printf("programa excedeu o limite de %d segundos\n", delay);
        kill(pid, SIGKILL);
    }
    return 0;
}
```

Primeiro, é associado o sinal *SIGCHLD*, que é o sinal que o processo pai recebe quando o filho termina. Esse sinal é associado a função *childhandler*, que será ainda definida.

Logo depois, há a divisão dos processos pai e filho, e o filho irá executando um loop eterno, enquanto o pai irá ler o delay que o usuário definiu, irá esperar esse tempo, e depois, enviar um sinal de morte para o filho. Note que se nesse meio tempo, o processo receber um sinal de que o filho acabou, o processo irá executar o *childhandler*.

```
void childhandler(int signo)
{
    int status;
    pid_t pid = wait(&status);
    printf("Child terminou em %d segundos com estado %d\n", pid, delay, status);
    exit(0);
}
```

A função *childhandler*, que é executada quando o filho acaba, simplesmente avisa que acabou e fecha o processo.

Resumindo, este programa irá executar o fork, e o filho irá ficar executando para sempre. Enquanto isso, o pai irá dormir por um tempo. Se ele receber um aviso avisando que o filho acabou, ele irá exibir que o filho acabou e sairá. Se não, ele irá cometer um ;filhocidio e sairá.

Testando trocando a função `for(EVER);` por `sleep(1);`

```
pi@raspberrypi:/mnt/rede/Daniel/rasc/sinais $ sudo gcc -o assassinato_evitado filhocidio.c
pi@raspberrypi:/mnt/rede/Daniel/rasc/sinais $ ./assassinato_evitado 3
Child 17686 terminou em 3 segundos com estado 0
pi@raspberrypi:/mnt/rede/Daniel/rasc/sinais $
```

Funcionou como deveria.

Questão 4:

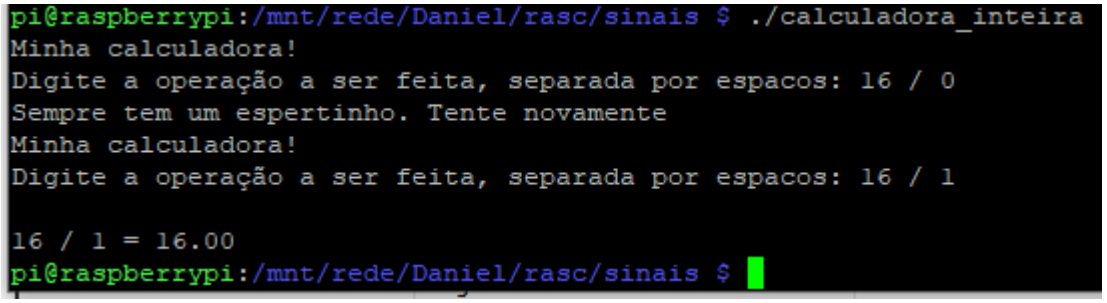
Código do programa original, usando números reais. Ele já está fazendo a captura do sinal SIGFPE:

```c

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

void zeroHandler(int signal);

int main() {
    float valor1, valor2, resposta;
    int v1, v2;
    char operacao;

    signal (SIGFPE, zeroHandler);

    printf("Minha calculadora!\n");
    printf("Digite a operação a ser feita, separada por espacos: ");
    scanf("%f %c %f", &valor1, &operacao, &valor2);
    // scanf("%d %c %d", &v1, &operacao, &v2);

    switch (operacao)
    {
        case '+':
            resposta = valor1 + valor2;
            printf("\n%0.2f + %0.2f = %0.2f\n", valor1, valor2, resposta);
            // resposta = v1 + v2;
            // printf("\n%d + %d = %0.2f\n", v1, v2, resposta);
            break;

        case '-':
            resposta = valor1 - valor2;
            printf("\n%0.2f - %0.2f = %0.2f\n", valor1, valor2, resposta);
            // resposta = v1 - v2;
            // printf("\n%d - %d = %0.2f\n", v1, v2, resposta);
            break;

        case '*':
            resposta = valor1 * valor2;
            printf("\n%0.2f * %0.2f = %0.2f\n", valor1, valor2, resposta);
            // resposta = v1 * v2;
            // printf("\n%d * %d = %0.2f\n", v1, v2, resposta);
            break;
        case '/':
            resposta = valor1 / valor2;
            printf("\n%0.2f / %0.2f = %0.2f\n", valor1, valor2, resposta);
            // resposta = v1 / v2;
            // printf("\n%d / %d = %0.2f\n", v1, v2, resposta);
            break;
    }
    return 0;
}

void zeroHandler(int signal) {
    printf("Sempre tem um espertinho. Tente novamente\n");
    main();
    exit(0);
}
```

```
pi@raspberrypi:/mnt/rede/Daniel/rasc/sinais $ ./calculadora
Minha calculadora!
Digite a operação a ser feita, separada por espacos: 16 / 0

16.00 / 0.00 = inf
```

Não houve o erro de divisão por zero, pois ao utilizar números reais, eles tem representação de infinito, portanto nem precisava da captura de erro de divisão por zero.

Porém, ao trocar de números reais para inteiros (no caso, tirar os comentários do código exibido acima e comentar a outra parte):



O código funcionou como esperado. Ele tratou o sinal de divisão por zero, chamou a função *zeroHandler,* que exibiu a mensagem e executou a main novamente.
