

RESOLUÇÃO DO CONCURSO 2

SISTEMAS OPERATIVOS 2024/2025

Feito por:
a83933 Rodrigo Linhas
2º Ano da Licenciatura de Engenharia Informática
(LEI)
Regente da UC: Amine Berqia

Índice

Introdução.....	3
EX 1.....	4
1).....	4
2).....	5
3).....	6
EX 2.....	8
1).....	8
2).....	9
EX 3.....	11
1).....	11
2).....	12
3).....	13
EX 4.....	15
1).....	15
2).....	16
Referencias bibliográficas.....	17

Introdução

Este trabalho tem o intuito de demonstrar a minha resolução perante os exercícios propostos no Concurso 1, juntamente com a(s) referencia(s) bibliográfica(s) consultada(s) e provas que funciona. Claramente que existem varias maneiras de solucionar os exercícios, no entanto com este documento demonstra como eu resolvi.

As soluções foram desenvolvidas em linguagem C, testadas em ambiente Linux Mint e validadas com capturas de ecrã para garantir conformidade com os requisitos. A estrutura do documento segue a numeração dos exercícios, com explicações concisas, exemplos de código e saídas geradas.

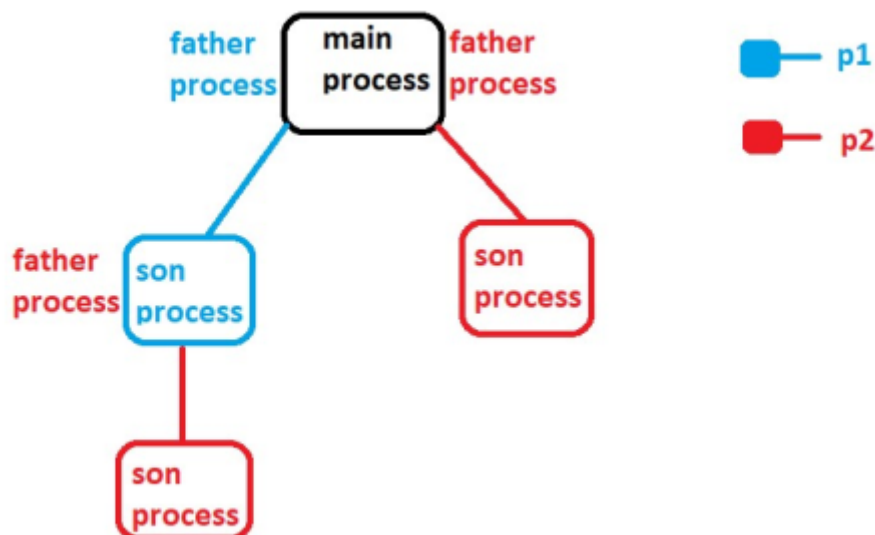
Palavras chave: processo, fork(), wait(), exec(), pipe()

EX 1

1)

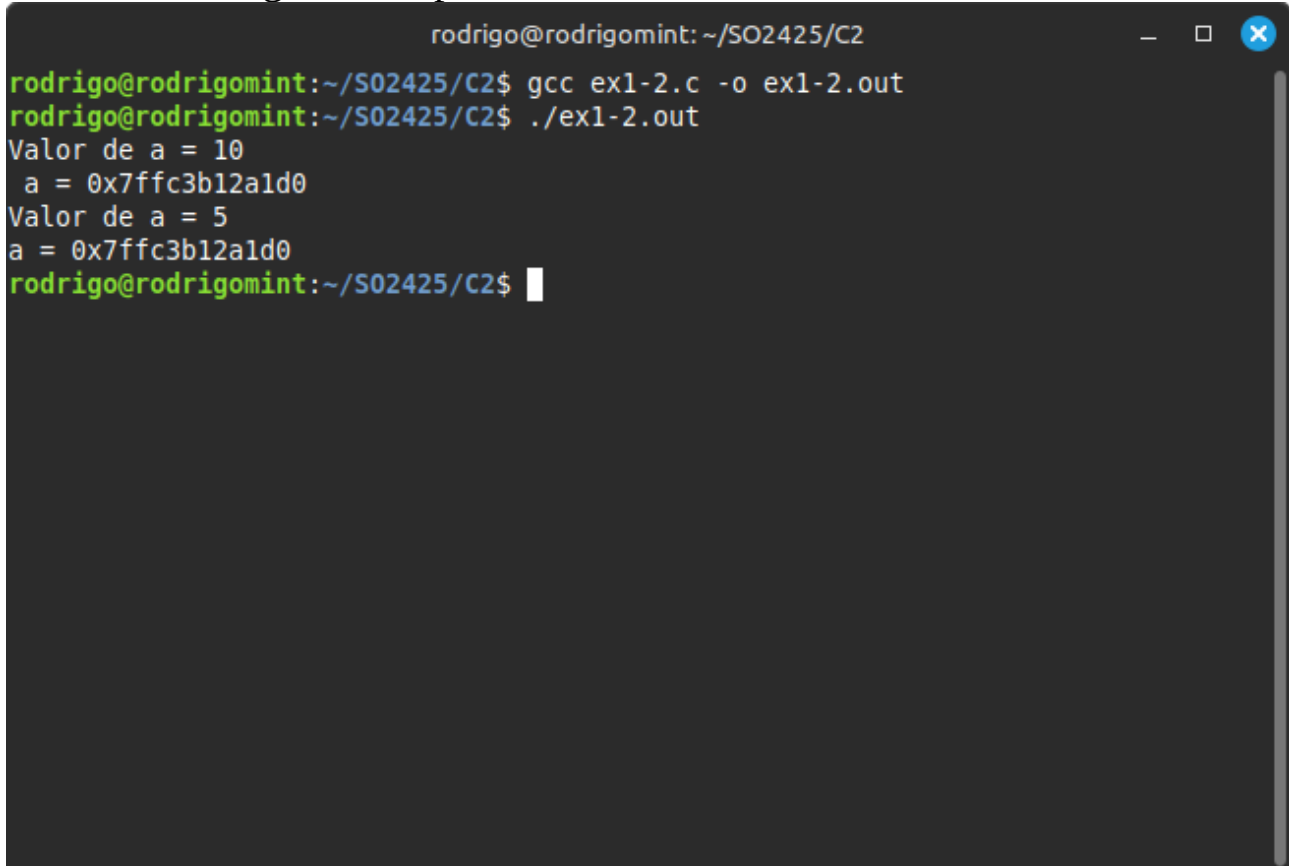
Para o código apresentado **são criados no total 4 processos**, sendo que quando invocamos a função *fork()* ele cria o processo pai e o processo filho, como invocamos 2 vezes eles vão se acumulando (2^2) e formando uma estrutura de hierárquica e mantendo dependência.

Eis uma visualização do mesmo:



2)

Para o programa apresentado após a compilação e execução do mesmo obtivemos o seguinte output:



```
rodrigo@rodrigomint: ~/SO2425/C2
rodrigo@rodrigomint:~/SO2425/C2$ gcc ex1-2.c -o ex1-2.out
rodrigo@rodrigomint:~/SO2425/C2$ ./ex1-2.out
Valor de a = 10
a = 0x7ffc3b12a1d0
Valor de a = 5
a = 0x7ffc3b12a1d0
rodrigo@rodrigomint:~/SO2425/C2$
```

Isto implica que **fez primeiro o processo filho e depois o pai por causa da função `wait()`**. A *variável a* foi declarado como 5, podemos ver contem o mesmo endereço de memória no entanto tem valores diferentes. Isso acontece porque o processo filho mudou a variável e é independente do processo pai.

3)

a)

Após ter compilado o programa dado, eis o output exercido:

```

rodrigo@rodrigomint:~/S02425/C2$ gcc ex1-3-a.c -o ex1-3-a.out
rodrigo@rodrigomint:~/S02425/C2$ ./ex1-3-a.out
  PID TTY          TIME CMD
 2615 pts/0        00:00:00 bash
 4811 pts/0        00:00:00 ex1-3-a.out
 4812 pts/0        00:00:00 ex1-3-a.out
 4813 pts/0        00:00:00 ex1-3-a.out
 4814 pts/0        00:00:00 sh
 4815 pts/0        00:00:00 ex1-3-a.out
 4816 pts/0        00:00:00 sh
 4817 pts/0        00:00:00 ps
 4818 pts/0        00:00:00 ps
  PID TTY          TIME CMD
 2615 pts/0        00:00:00 bash
 4811 pts/0        00:00:00 ex1-3-a.out
 4812 pts/0        00:00:00 ex1-3-a.out
 4813 pts/0        00:00:00 ex1-3-a.out
 4814 pts/0        00:00:00 sh
 4815 pts/0        00:00:00 ex1-3-a.out
 4816 pts/0        00:00:00 sh
 4817 pts/0        00:00:00 ps
 4818 pts/0        00:00:00 ps
  PID TTY          TIME CMD
 2615 pts/0        00:00:00 bash
 4811 pts/0        00:00:00 ex1-3-a.out
 4813 pts/0        00:00:00 ex1-3-a.out
 4819 pts/0        00:00:00 sh
 4820 pts/0        00:00:00 sh
 4821 pts/0        00:00:00 ps
 4822 pts/0        00:00:00 ps
  PID TTY          TIME CMD
 2615 pts/0        00:00:00 bash
 4811 pts/0        00:00:00 ex1-3-a.out
 4813 pts/0        00:00:00 ex1-3-a.out
 4819 pts/0        00:00:00 sh
 4820 pts/0        00:00:00 sh
 4821 pts/0        00:00:00 ps
 4822 pts/0        00:00:00 ps
rodrigo@rodrigomint:~/S02425/C2$ █

```

No entanto encontrarmo-nos na presença de um erro neste código, o **`wait(&pid)`**. Porque **espera apenas por um processo filho**, enquanto a expressão **`pid_t pid = (fork() && (fork() || fork()))`** cria 4 processos no

total. Os restantes filhos tornam-se *zombies* ou executam `system("ps")` simultaneamente, gerando saídas repetidas e inconsistências de dados. A lógica dos operadores `&&` e `||` multiplica os forks de forma não intuitiva, e a falta de `wait` para todos os filhos causa execução descontrolada.

b)

Para a correção do programa anterior foi feito o seguinte:

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main(void) {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0) {
        //filho
        exit(0);
    } else {
        while (wait(&status) > 0); // Espera por todos os filhos
    }
    system("ps");
    exit(0);
}

```

agora o output fica mais consistente:

```

rodrigo@rodrigomint:~/S02425/C2$ gcc ex1-3-b.c -o ex1-3-b.out
rodrigo@rodrigomint:~/S02425/C2$ ./ex1-3-b.out
  PID TTY          TIME CMD
 2615 pts/0        00:00:00 bash
 5709 pts/0        00:00:00 ex1-3-b.out
 5711 pts/0        00:00:00 sh
 5712 pts/0        00:00:00 ps
rodrigo@rodrigomint:~/S02425/C2$

```

EX 2

1)

Para este exercício criei o seguinte programa:

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <sys/wait.h>
4
5  int main() {
6      pid_t pid = fork();
7
8      if (pid > 0) { // Pai
9          for (int i = 0; i < 3; i++)
10             printf("Eu sou o pai, minha identificação é %d\n", getpid());
11             wait(NULL);
12     } else if (pid == 0) { // Filho
13         for (int i = 0; i < 5; i++)
14             printf("Eu sou o filho, meu pai é %d\n", getppid());
15     }
16     return 0;
17 }
18
```

e o output obtido foi este:

```
rodrigo@rodrigomint:~/S02425/C2$ gcc ex2-1.c -o ex2-1.out
rodrigo@rodrigomint:~/S02425/C2$ ./ex2-1.out
Eu sou o pai, minha identificação é 2431
Eu sou o pai, minha identificação é 2431
Eu sou o pai, minha identificação é 2431
Eu sou o filho, meu pai é 2431
Eu sou o filho, meu pai é 2431
Eu sou o filho, meu pai é 2431
Eu sou o filho, meu pai é 2431
Eu sou o filho, meu pai é 2431
```


2)

Para as especificações dadas, peguei no programa anterior e fiz as seguintes alterações:

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

int main(void){
    pid_t pid = fork();
    //cria 3 filhos
    pid_t pid1,pid2;
    if(pid > 0){
        pid1 = fork();
        if(pid1 > 0){
            pid1 = fork();
            if(pid1 > 0){
                pid1 = fork();
            }
        }
    }
    //cria 2 filhos do 1 filho
    else if(pid == 0){
        pid2 = fork();
        if(pid2 > 0){
            pid2 = fork();
        }
    }
    //se é pai ou lo filho
    if(pid > 0 && pid1 > 0 || pid == 0 && pid2 > 0){
        //pai é impresso primeiro (sleep)
        if(pid == 0){
            sleep(1);
        }
        for(int i = 0; i < 3; i++){
            printf("Eu sou o pai, a minha identificação é %ld\n", (long) getpid());
        }
    }
    else{
        sleep(2);
        //filhos do pai são impressos (sleep)
        if(pid == 0){
            sleep(1);
        }
        for(int i = 0; i < 5; i++){
            printf("Eu sou o filho o meu pai é %ld\n", (long) getppid());
        }
    }
    wait(&pid);
    wait(&pid);
    wait(&pid);
    wait(&pid);
    exit(0);
}
```

o output adquirido foi este (VER NA PRÓXIMA PAGINA):

```
rodrigo@rodrigomint:~/S02425/C2$ gcc ex2-2.c -o ex2-2.out
rodrigo@rodrigomint:~/S02425/C2$ ./ex2-2.out
Eu sou o pai, a minha identificação é 5372
Eu sou o pai, a minha identificação é 5372
Eu sou o pai, a minha identificação é 5372
Eu sou o pai, a minha identificação é 5373
Eu sou o pai, a minha identificação é 5373
Eu sou o pai, a minha identificação é 5373
Eu sou o filho o meu pai é 5372
Eu sou o filho o meu pai é 5372
Eu sou o filho o meu pai é 5372
Eu sou o filho o meu pai é 5372
Eu sou o filho o meu pai é 5372
Eu sou o filho o meu pai é 5372
Eu sou o filho o meu pai é 5372
Eu sou o filho o meu pai é 5372
Eu sou o filho o meu pai é 5372
Eu sou o filho o meu pai é 5372
Eu sou o filho o meu pai é 5372
Eu sou o filho o meu pai é 5372
Eu sou o filho o meu pai é 5372
Eu sou o filho o meu pai é 5373
Eu sou o filho o meu pai é 5373
Eu sou o filho o meu pai é 5373
Eu sou o filho o meu pai é 5373
Eu sou o filho o meu pai é 5373
Eu sou o filho o meu pai é 5373
Eu sou o filho o meu pai é 5373
Eu sou o filho o meu pai é 5373
Eu sou o filho o meu pai é 5373
Eu sou o filho o meu pai é 5373
rodrigo@rodrigomint:~/S02425/C2$
```

EX 3

1)

i)

Para os comandos *who & ps & ls -l* os mesmos **são executados em paralelo**, ou por outras palavras o terminal não espera que um comando termine para iniciar o próximo.

A saída pode aparecer misturada, pois os processos competem pelo terminal, na alínea 2 contem uma demonstração visual.

ii)

Para os comandos *who ; ps ; ls -l* os mesmos **são executados sequencialmente**, ou por outras palavras, o terminal aguarda cada comando terminar antes de iniciar o próximo.

A saída é ordenada: primeiro *who*, depois *ps*, por fim *ls -l*, na alínea 3 contem uma demonstração visual

2)

Para o problema proposto fiz o seguinte programa:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main(void) {
    if (fork() == 0) execlp("who", "who", NULL);
    if (fork() == 0) execlp("ps", "ps", NULL);
    if (fork() == 0) execlp("ls", "ls", "-l", NULL);
    while (wait(NULL) > 0);
    return 0;
}
```

Obtendo o output:

```
rodrigo@rodrigomint:~/S02425/C2$ gcc ex3-2.c -o ex3-2.out
rodrigo@rodrigomint:~/S02425/C2$ ./ex3-2.out
rodrigo tty7          2025-03-18 18:51 (:0)
total 1132
-rw-rw-r-- 1 rodrigo rodrigo 597370 mar 18 11:27 C2-0S-24-25.pdf
-rw-rw-r-- 1 rodrigo rodrigo 419749 mar 18 15:55 concurso_FINAL.odt
-rw-rw-r-- 1 rodrigo rodrigo   206 mar 18 13:18 ex1-1.c
-rw-rw-r-- 1 rodrigo rodrigo   497 mar 18 13:24 ex1-2.c
-rwxrwxr-x 1 rodrigo rodrigo 16088 mar 18 13:20 ex1-2.out
-rw-rw-r-- 1 rodrigo rodrigo   235 mar 18 13:31 ex1-3-a.c
-rwxrwxr-x 1 rodrigo rodrigo 16088 mar 18 13:40 ex1-3-a.out
-rw-rw-r-- 1 rodrigo rodrigo   351 mar 18 15:37 ex1-3-b.c
-rwxrwxr-x 1 rodrigo rodrigo 16088 mar 18 13:56 ex1-3-b.out
-rw-rw-r-- 1 rodrigo rodrigo   447 mar 18 15:44 ex2-1.c
-rwxrwxr-x 1 rodrigo rodrigo 16128 mar 18 15:40 ex2-1.out
-rw-rw-r-- 1 rodrigo rodrigo   933 mar 18 19:32 ex2-2.c
-rwxrwxr-x 1 rodrigo rodrigo 16216 mar 18 19:24 ex2-2.out
-rw-rw-r-- 1 rodrigo rodrigo   285 mar 18 19:35 ex3-2.c
-rwxrwxr-x 1 rodrigo rodrigo 16048 mar 18 19:37 ex3-2.out
-rw-rw-r-- 1 rodrigo rodrigo   349 mar 18 11:46 ex3-3.c
-rw-rw-r-- 1 rodrigo rodrigo   335 mar 18 11:46 ex4-a.c
-rw-rw-r-- 1 rodrigo rodrigo   397 mar 18 11:47 ex4-b.c
prw-rw-r-- 1 rodrigo rodrigo    0 mar 18 12:56 pipe-criado-com-mkfifo
  PID TTY          TIME CMD
  4539 pts/0        00:00:00 bash
  6063 pts/0        00:00:00 ex3-2.out
  6065 pts/0        00:00:00 ps
rodrigo@rodrigomint:~/S02425/C2$
```

3)

Para o problema proposto fiz o seguinte programa:

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

int main(){
    pid_t p1 = fork();
    if(p1 == 0){
        pid_t p2 = fork();
        if(p2 == 0){
            execlp("who", "who", NULL);
        }
        else if(p2 > 0){
            wait(&p2);
            execlp("ps", "ps", NULL);
        }
    }
    else if(p1 > 0){
        wait(&p1);
        execlp("ls", "ls", "-l", NULL);
    }

    exit(0);
}
```

Obtendo o seguinte output (VER NA PRÓXIMA PAGINA):

```
rodrigo@rodrigomint:~/S02425/C2$ gcc ex3-3.c -o ex3-3.out
rodrigo@rodrigomint:~/S02425/C2$ ./ex3-3.out
rodrigo  tty7          2025-03-18 18:51 (:0)
  PID TTY          TIME CMD
  4539 pts/0        00:00:00 bash
  6293 pts/0        00:00:00 ex3-3.out
  6294 pts/0        00:00:00 ps
total 1148
-rw-rw-r-- 1 rodrigo rodrigo 597370 mar 18 11:27 C2-0S-24-25.pdf
-rw-rw-r-- 1 rodrigo rodrigo 419749 mar 18 15:55 concurso_FINAL.odt
-rw-rw-r-- 1 rodrigo rodrigo   206 mar 18 13:18 ex1-1.c
-rw-rw-r-- 1 rodrigo rodrigo   497 mar 18 13:24 ex1-2.c
-rwxrwxr-x 1 rodrigo rodrigo 16088 mar 18 13:20 ex1-2.out
-rw-rw-r-- 1 rodrigo rodrigo   235 mar 18 13:31 ex1-3-a.c
-rwxrwxr-x 1 rodrigo rodrigo 16088 mar 18 13:40 ex1-3-a.out
-rw-rw-r-- 1 rodrigo rodrigo   351 mar 18 15:37 ex1-3-b.c
-rwxrwxr-x 1 rodrigo rodrigo 16088 mar 18 13:56 ex1-3-b.out
-rw-rw-r-- 1 rodrigo rodrigo   447 mar 18 15:44 ex2-1.c
-rwxrwxr-x 1 rodrigo rodrigo 16128 mar 18 15:40 ex2-1.out
-rw-rw-r-- 1 rodrigo rodrigo   933 mar 18 19:32 ex2-2.c
-rwxrwxr-x 1 rodrigo rodrigo 16216 mar 18 19:24 ex2-2.out
-rw-rw-r-- 1 rodrigo rodrigo   285 mar 18 19:35 ex3-2.c
-rwxrwxr-x 1 rodrigo rodrigo 16048 mar 18 19:37 ex3-2.out
-rw-rw-r-- 1 rodrigo rodrigo   385 mar 18 19:42 ex3-3.c
-rwxrwxr-x 1 rodrigo rodrigo 16088 mar 18 19:42 ex3-3.out
-rw-rw-r-- 1 rodrigo rodrigo   335 mar 18 11:46 ex4-a.c
-rw-rw-r-- 1 rodrigo rodrigo   397 mar 18 11:47 ex4-b.c
prw-rw-r-- 1 rodrigo rodrigo    0 mar 18 12:56 pipe-criado-com-mkfifo
rodrigo@rodrigomint:~/S02425/C2$
```

EX 4

1)

Para o problema proposto fiz o seguinte programa:

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>

int main(void) {
    int fd[2];
    pipe(fd);
    char msg[] = "Hello, it's me";

    write(fd[1], msg, strlen(msg)+1);
    close(fd[1]);

    char buf[50];
    read(fd[0], buf, sizeof(buf));
    printf("String lida: %s\n", buf);
    return 0;
}
```

Obtendo o seguinte output:

```
rodrigo@rodrigomint:~/S02425/C2$ gcc ex4-a.c -o ex4-a.out
rodrigo@rodrigomint:~/S02425/C2$ ./ex4-a.out
String lida: Hello, it's me
rodrigo@rodrigomint:~/S02425/C2$
```

2)

Para o problema proposto fiz o seguinte programa:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main(void) {
    int fd[2];
    pipe(fd);
    pid_t pid = fork();
    if (pid == 0) {
        dup2(fd[1], STDOUT_FILENO);
        close(fd[0]);
        execlp("echo", "echo", "Ola Hi", NULL);
    } else {
        dup2(fd[0], STDIN_FILENO);
        close(fd[1]);
        execlp("wc", "wc", NULL);
    }
    return 0;
}
```

Obtendo o seguinte output:

```
rodrigo@rodrigomint:~/S02425/C2$ gcc ex4-b.c -o ex4-b.out
rodrigo@rodrigomint:~/S02425/C2$ ./ex4-b.out
1      2      7
rodrigo@rodrigomint:~/S02425/C2$
```


Referencias bibliográficas

Eis todos os sites que eu consultei na resolução deste concurso:

- <https://www.geeksforgeeks.org/>
- <https://stackoverflow.com/>
- [https://www.tutorialspoint.com/
inter_process_communication/
inter_process_communication_pipes.htm](https://www.tutorialspoint.com/inter_process_communication/inter_process_communication_pipes.htm)
- [https://pubs.opengroup.org/onlinepubs/9699919799/
functions/](https://pubs.opengroup.org/onlinepubs/9699919799/functions/)