Sistemas Operativos

Concurso 2 – Processos

67990 – José Diogo Ferrás

Ex. I

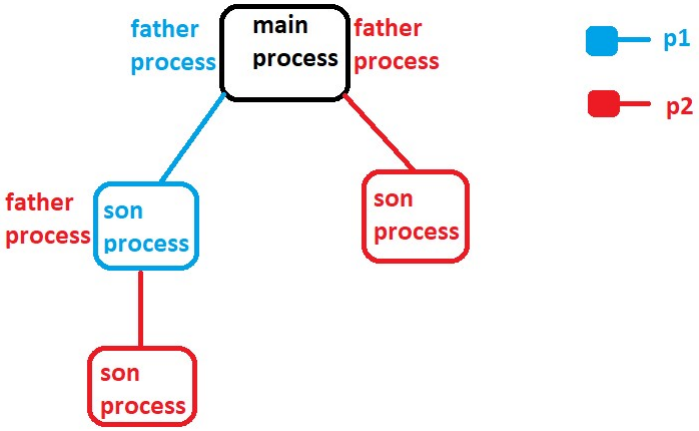
1)

```c
#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,p2;
 p1 = fork();
 p2 = fork();
 exit(0);
}
```

Este programa irá criar, no total, 4 processos (2^2).

Aqui está um pequeno esquema que representa os processos criados por este programa e a relação entre eles.



2)

Código:

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

int main(){

    int a = 5;
    pid_t pid;
    //if pid != 0 (when its the father)
    if(pid=fork()){
        wait(&pid);
        printf("Valor de a = %d\n",a);
        printf("a = %p\n",&a);
    }
    //if pid == 0 (when its the son)
    else{
        a = 10;
        printf("Valor de a = %d\n",a);
        printf("a = %p\n",&a);
    }
    exit(0);

}
```

I/O:

```
zediogo@ubuntu:~/Documents/c2$ ./a.out
Valor de a = 10
a = 0x7ffd229117d0
Valor de a = 5
a = 0x7ffd229117d0
```

Como podemos observar no I/O, o código executado pelo processo filho é executado antes do código do processo pai.

Isto acontece pois o processo pai fica à espera que o processo filho acabe graças ao comando wait().

3)

a)

Código:

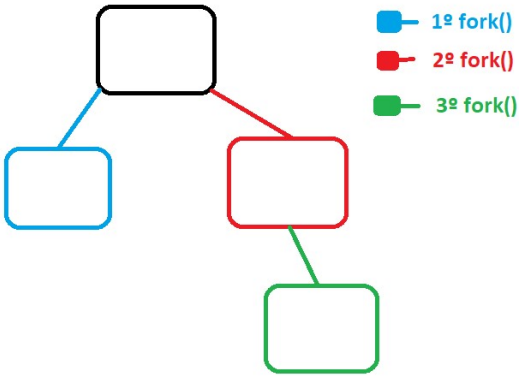
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(){
    pid_t pid = (fork() && (fork() || fork()));
    wait(&pid);
    system("ps");
    exit(0);
}
```

I/O:

```
zediogo@ubuntu:~/Documents/c2$ ./a.out
    PID TTY          TIME CMD
    PID TTY          TIME CMD
   2625 pts/0    00:00:00 bash
   3818 pts/0    00:00:00 a.out
   3819 pts/0    00:00:00 a.out
   3820 pts/0    00:00:00 a.out
   3821 pts/0    00:00:00 a.out
   3822 pts/0    00:00:00 sh
   3823 pts/0    00:00:00 sh
   3824 pts/0    00:00:00 ps
   3825 pts/0    00:00:00 ps
   2625 pts/0    00:00:00 bash
   3818 pts/0    00:00:00 a.out
   3820 pts/0    00:00:00 a.out
   3821 pts/0    00:00:00 a.out
   3823 pts/0    00:00:00 sh
   3825 pts/0    00:00:00 ps
    PID TTY          TIME CMD
   2625 pts/0    00:00:00 bash
   3818 pts/0    00:00:00 a.out
   3820 pts/0    00:00:00 a.out
   3826 pts/0    00:00:00 sh
   3827 pts/0    00:00:00 ps
   3828 pts/0    00:00:00 sh
zediogo@ubuntu:~/Documents/c2$     PID TTY          TIME CMD
   2625 pts/0    00:00:00 bash
   3820 pts/0    00:00:00 a.out
   3828 pts/0    00:00:00 sh
   3829 pts/0    00:00:00 ps
```

Aqui está um esquema dos processos criados pelo programa:



O que está acontecer de errado nesta execução do programa é que como o wait(&pid) apenas espera que 1 dos filhos acabe para poder continuar, é possível que, quando o processo azul acaba, o wait(&pid) do processo pai acaba, e continua a executar, acabando o programa, fazendo com que o processo vermelho fique sem processo pai, sendo-lhe atribuído um aleatório pelo sistema operativo.

Consegui chegar a esta conclusão graças a um teste com o seguinte printf:

```
wait(&pid);
printf("%ld% ld\n", (long)getpid(), (long)getppid());
```

Com o qual obtive os seguintes resultados:

```
zediogo@ubuntu:~/Documents/c2$ 4602 1501
./a.out
4605 4604
4607 4606
4604 2625
zediogo@ubuntu:~/Documents/c2$ 4606 1501
```

Nota: Poderia até acontecer que o processo azul e o processo pai acaba antes do verde sequer executar, o que fará com que o output destes seja escrito para a consola depois de o programa “origem” terminar.

b)

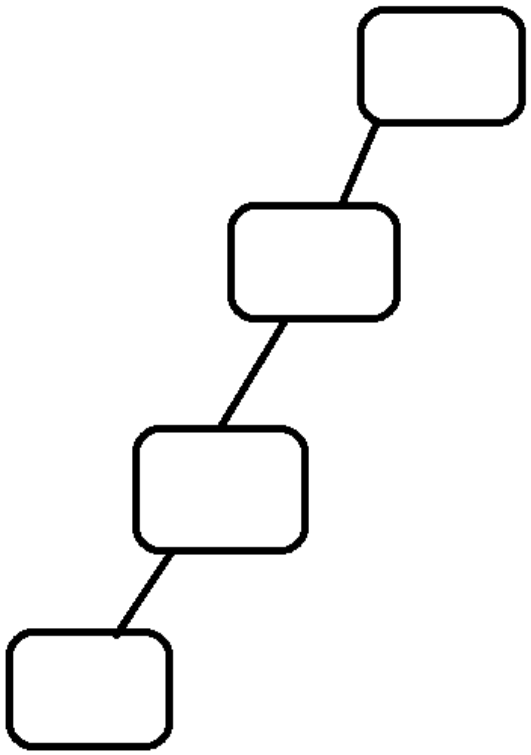
Supondo que a ideia seria utilizar 4 vezes, independentemente o comando ps, podemos fazer o seguinte:

Código:

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

int main(){
    pid_t pid = fork();
    if(pid == 0){
        pid = fork();
        if(pid == 0){
            pid = fork();
        }
    }
    if(pid > 0){
        wait(&pid);
    }
    //printf("%ld% ld\n", (long)getpid(), (long)getppid());
    system("ps");
    exit(0);
}
```

Isto irá criar um esquema de forks deste género:



Fazendo como que cada processo espere individualmente pelo seu filho, e executando os processos um de cada vez, em vez de em simultâneo, como acontecia no programa anteriormente.

I/O:

```
zediogo@ubuntu:~/Documents/c2$ ./a.out
    PID TTY          TIME CMD
   2625 pts/0    00:00:00 bash
   6952 pts/0    00:00:00 a.out
   6953 pts/0    00:00:00 a.out
   6954 pts/0    00:00:00 a.out
   6955 pts/0    00:00:00 a.out
   6956 pts/0    00:00:00 sh
   6957 pts/0    00:00:00 ps
    PID TTY          TIME CMD
   2625 pts/0    00:00:00 bash
   6952 pts/0    00:00:00 a.out
   6953 pts/0    00:00:00 a.out
   6954 pts/0    00:00:00 a.out
   6958 pts/0    00:00:00 sh
   6959 pts/0    00:00:00 ps
    PID TTY          TIME CMD
   2625 pts/0    00:00:00 bash
   6952 pts/0    00:00:00 a.out
   6953 pts/0    00:00:00 a.out
   6960 pts/0    00:00:00 sh
   6961 pts/0    00:00:00 ps
    PID TTY          TIME CMD
   2625 pts/0    00:00:00 bash
   6952 pts/0    00:00:00 a.out
   6962 pts/0    00:00:00 sh
   6963 pts/0    00:00:00 ps
```

Ex II.

1)

Código:

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

int main(){
    pid_t pid = fork();
    if(pid > 0){
        for(int i = 0; i < 3; i++){
            printf("Eu sou o pai, a minha identificação é %ld\n", (long)getpid());
        }
    }
    else{
        sleep(1);
        for(int i = 0; i < 5; i++){
            printf("Eu sou o filho o meu pai é %ld\n", (long)getppid());
        }
    }
    wait(&pid);
    exit(0);
}
```

I/O:

```
zediogo@ubuntu:~/Documents/c2$ ./a.out
Eu sou o pai, a minha identificação é 7688
Eu sou o pai, a minha identificação é 7688
Eu sou o pai, a minha identificação é 7688
Eu sou o filho o meu pai é 7688
Eu sou o filho o meu pai é 7688
Eu sou o filho o meu pai é 7688
Eu sou o filho o meu pai é 7688
Eu sou o filho o meu pai é 7688
```

2)

Código:

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

int main(){
    pid_t pid = fork();
    //create 3 extra children to root
    pid_t pid1,pid2;
    if(pid > 0){
        pid1 = fork();
        if(pid1 > 0){
            pid1 = fork();
            if(pid1 > 0){
                pid1 = fork();
            }
        }
    }
    //create 2 extra children to first child
    else if(pid == 0){
        pid2 = fork();
        if(pid2 > 0){
            pid2 = fork();
        }
    }
    //if its the root or the first children
    if(pid > 0 && pid1 > 0 || pid == 0 && pid2 > 0){
        //sleep so that root is printed first
        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);
        //sleep so that sons of root are printed first
        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);
}
```

I/O:

```
zed1ogo@ubuntu:~/Documents/c2$ ./a.out
Eu sou o pai, a minha identificação é 11220
Eu sou o pai, a minha identificação é 11220
Eu sou o pai, a minha identificação é 11220
Eu sou o pai, a minha identificação é 11221
Eu sou o pai, a minha identificação é 11221
Eu sou o pai, a minha identificação é 11221
Eu sou o filho o meu pai é 11220
Eu sou o filho o meu pai é 11220
Eu sou o filho o meu pai é 11220
Eu sou o filho o meu pai é 11220
Eu sou o filho o meu pai é 11220
Eu sou o filho o meu pai é 11220
Eu sou o filho o meu pai é 11220
Eu sou o filho o meu pai é 11220
Eu sou o filho o meu pai é 11220
Eu sou o filho o meu pai é 11220
Eu sou o filho o meu pai é 11220
Eu sou o filho o meu pai é 11220
Eu sou o filho o meu pai é 11220
Eu sou o filho o meu pai é 11220
Eu sou o filho o meu pai é 11220
Eu sou o filho o meu pai é 11221
Eu sou o filho o meu pai é 11221
Eu sou o filho o meu pai é 11221
Eu sou o filho o meu pai é 11221
Eu sou o filho o meu pai é 11221
Eu sou o filho o meu pai é 11221
Eu sou o filho o meu pai é 11221
Eu sou o filho o meu pai é 11221
Eu sou o filho o meu pai é 11221
```

Ex III.

1) A diferença entre o comando “who & ps & ls –l” e o comando “who ; ps ; ls –l” são os operandos “&” e “;".
   O operando “&” faz com que os 3 comandos sejam executados consecutivamente, enquanto que o operando “;” faz com que os comandos sejam executados sequencialmente, esperando pela execução do atual para executar o próximo.

2)

Código:

```c
#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("ls","ls","-l",NULL);
        }
        else if(p2 == 0){
            execlp("ps","ps",NULL);
        }
    }
    else if(p1 == 0){
        execlp("who","who",NULL);
    }
    exit(0);
}
```

I/O:

```
zediogo@ubuntu:~/Documents/c2$ ./a.out
zediogo  :0           2024-03-09 06:32 (:0)
total 92
-rwxrwxr-x 1 zediogo zediogo 16784 Mar  9 08:47 a.out
-rw-rw-r-- 1 zediogo zediogo   190 Mar  6 08:41 ex1_1.c
-rw-rw-r-- 1 zediogo zediogo   438 Mar  6 08:41 ex1_2.c
-rw-rw-r-- 1 zediogo zediogo   343 Mar  9 07:31 ex1_3.c
-rw-rw-r-- 1 zediogo zediogo   431 Mar  9 07:41 ex2_1.c
-rw-rw-r-- 1 zediogo zediogo   988 Mar  9 08:25 ex2_2.c
-rw-rw-r-- 1 zediogo zediogo   357 Mar  9 08:48 ex3_2.c
-rw-rw-r-- 1 zediogo zediogo   859 Mar  5 09:46 exec-example
-rw-rw-r-- 1 zediogo zediogo   193 Mar  5 09:51 fork_example0.c
-rw-rw-r-- 1 zediogo zediogo   150 Mar  5 09:51 fork_example1.c
-rw-rw-r-- 1 zediogo zediogo   539 Mar  5 09:52 fork_example2.c
-rw-rw-r-- 1 zediogo zediogo   371 Mar  5 09:52 fork_example3.c
-rw-rw-r-- 1 zediogo zediogo   234 Mar  5 09:53 fork_example4.c
-rw-rw-r-- 1 zediogo zediogo   231 Mar  5 09:53 fork_example5.c
-rw-rw-r-- 1 zediogo zediogo   637 Mar  5 09:53 fork_example6.c
-rw-rw-r-- 1 zediogo zediogo   407 Mar  5 09:54 fork_example7.c
-rw-rw-r-- 1 zediogo zediogo   477 Mar  5 09:54 fork_example8.c
-rw-rw-r-- 1 zediogo zediogo   394 Mar  9 07:51 forkTests.c
-rw-rw-r-- 1 zediogo zediogo  3608 Mar  5 09:46 fork-wait-examples
zediogo@ubuntu:~/Documents/c2$     PID TTY          TIME CMD
   2625 pts/0    00:00:00 bash
  13098 pts/0    00:00:00 ps
```

3)

Código:

```c
#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);
}
```

I/O:

```
zediogo@ubuntu:~/Documents/c2$ ./a.out
zediogo  :0           2024-03-09 06:32 (:0)
    PID TTY          TIME CMD
   2625 pts/0    00:00:00 bash
  13544 pts/0    00:00:00 a.out
  13545 pts/0    00:00:00 ps
total 96
-rwxrwxr-x 1 zediogo zediogo 16824 Mar  9 08:53 a.out
-rw-rw-r-- 1 zediogo zediogo   190 Mar  6 08:41 ex1_1.c
-rw-rw-r-- 1 zediogo zediogo   438 Mar  6 08:41 ex1_2.c
-rw-rw-r-- 1 zediogo zediogo   343 Mar  9 07:31 ex1_3.c
-rw-rw-r-- 1 zediogo zediogo   431 Mar  9 07:41 ex2_1.c
-rw-rw-r-- 1 zediogo zediogo   988 Mar  9 08:25 ex2_2.c
-rw-rw-r-- 1 zediogo zediogo   359 Mar  9 08:51 ex3_2.c
-rw-rw-r-- 1 zediogo zediogo   385 Mar  9 08:53 ex3_3.c
-rw-rw-r-- 1 zediogo zediogo   859 Mar  5 09:46 exec-example
-rw-rw-r-- 1 zediogo zediogo   193 Mar  5 09:51 fork_example0.c
-rw-rw-r-- 1 zediogo zediogo   150 Mar  5 09:51 fork_example1.c
-rw-rw-r-- 1 zediogo zediogo   539 Mar  5 09:52 fork_example2.c
-rw-rw-r-- 1 zediogo zediogo   371 Mar  5 09:52 fork_example3.c
-rw-rw-r-- 1 zediogo zediogo   234 Mar  5 09:53 fork_example4.c
-rw-rw-r-- 1 zediogo zediogo   231 Mar  5 09:53 fork_example5.c
-rw-rw-r-- 1 zediogo zediogo   637 Mar  5 09:53 fork_example6.c
-rw-rw-r-- 1 zediogo zediogo   407 Mar  5 09:54 fork_example7.c
-rw-rw-r-- 1 zediogo zediogo   477 Mar  5 09:54 fork_example8.c
-rw-rw-r-- 1 zediogo zediogo   394 Mar  9 07:51 forkTests.c
-rw-rw-r-- 1 zediogo zediogo  3608 Mar  5 09:46 fork-wait-examples
```

Ex IV:

1)

Código:

```c
int main(){
    int fd[2];
    pipe(fd);
    pid_t id = fork();

    if (id == 0) {
        // Child process
        char * w = "Hello, it's me";
        close(fd[0]);
        if (write(fd[1], &w, sizeof(int)) == -1) {
            printf("An error ocurred with writing to the pipe\n");
            return 3;
        }
        close(fd[1]);
    } else {
        // Parent process
        close(fd[1]);
        char * w;
        if (read(fd[0], &w, sizeof(int)) == -1) {
            printf("An error ocurred with reading from the pipe\n");
            return 4;
        }
        printf("%s\n", w);
        close(fd[0]);
    }
}
```

I/O:

```
zediogo@ubuntu:~/Documents/c2$ ./a.out
Hello, it's me
```

2)

Código:

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

int main(){
    int fd[2];
    pipe(fd);
    pid_t id = fork();

    if (id == 0) {
        // Child process
        close(fd[0]);
        dup2(fd[1], STDOUT_FILENO);
        execlp("echo", "echo", "Ola Hi", NULL);
        close(fd[1]);
    } else {
        // Parent process
            close(fd[1]);
            dup2(fd[0], STDIN_FILENO);
            execlp("wc", "wc", NULL);
            close(fd[0]);
    }
    exit(0);
}
```

I/O:

```
zediogo@ubuntu:~/Documents/c2$ ./a.out
       1       2       7
```