

# Sistemas Operativos

## Licenciatura em Engenharia Informática

### Soluções dos exercícios de criação de processos – fork()

- 1) Quantos processos são criados no seguinte programa?

```
int main() {  
    fork();  
    fork();  
}
```

**4 Processos**

- 2) Desenhe a árvore de criação de processos do seguinte programa, e diga quantos processos são criados ao todo.

```
int main() {  
    fork();  
    fork();  
    if (fork() == 0) {  
        if (fork() == 0) {  
            fork();  
        }  
    }  
}
```

**16 Processos**

Nas linhas *if*, o programa executa sempre primeiro os *forks* e só depois vai validar a igualdade. Nestes casos, apenas os processos filhos executam as instruções após os *ifs*.

- 3) Quais são os outputs nas linhas “X” e “Y”? Assuma que a função *wait(NULL)* permite que o processo pai espere pelo fim do processo filho.

```
int main() {  
    int i;  
    int size = 5;  
    int nums[size] = {0, 1, 2, 3, 4};  
  
    int pid = fork();  
    if (pid == 0) {  
        for (i=0; i<size; i++) {  
            nums[i] = nums[i] * -1;  
            printf("Child: %d", nums[i]);           // Linha X  
        }  
    }  
    else {  
        wait(NULL);  
        for (i=0; i<size; i++) {  
            printf("Parent: %d", nums[i]);         // Linha Y  
        }  
    }  
}
```

**Child: 0 -1 -2 -3 -4**

**Parent: 0 1 2 3 4**

Os processos estão isolados em termos de memória, logo a alteração do array no processo filho não afecta o array no pai.

4) Quantos processos são criados no seguinte programa?

```
int main() {  
    for (int i=0; i<4; i++)  
        fork();  
}
```

**16 Processos**

Técnica muito usada para criar  $n$  processos que serão depois usados como *workers*.

5) Quantos processos são criados no seguinte programa?

```
int main() {  
    for (int i=0; i<4; i++) {  
        fork();  
        exit(0);  
    }  
}
```

**2 Processos**

Após o primeiro fork, ambos os processos fazem `exit()`

6) Quantos processos são criados no seguinte programa?

```
int main() {  
    for (int i=0; i<4; i++) {  
        if (fork() == 0)  
            exit(0);  
    }  
}
```

**5 Processos.**

O processo pai vai criando processos filhos que vão terminando.

7) A função `getpid()` permite obter o *process id* de um processo. Assumindo que os pids do processo pai e do processo filho são respectivamente 2600 e 2603, identifique os valores escritos nas linhas A, B, C e D.

```
int main() {  
    int pid1, pid2;  
  
    pid1 = fork();  
    if (pid1 == 0) {  
        pid2 = getpid();  
        printf("child: pid1=%d", pid1);           // A  
        printf("child: pid2=%d", pid2);           // B  
    } else {  
        pid2 = getpid();  
        printf("parent: pid1=%d", pid1);           // C  
        printf("parent: pid2=%d", pid2);           // D  
        wait(NULL);  
    }  
}
```

**A) child: pid1=0**

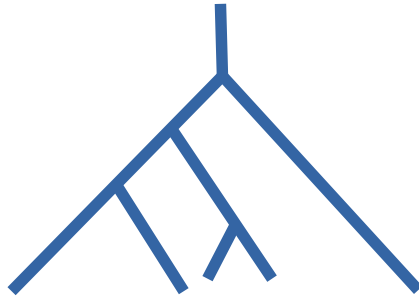
**B) child: pid2=2603**

**C) parent: pid1=2603**

**D) parent: pid2=2600**

O `fork()` cria 1 processo filho e nesse processo a variável de retorno (`pid1`) fica com o valor 0. No processo pai, essa variável fica com o ID do filho.

- 8) Escreva um programa em C que crie a seguinte árvore de processos. Assuma que sempre que é feito um *fork()*, o processo filho é sempre o processo que segue para a direita.



Assume-se que os processos filhos são os que seguem sempre para a direita após o *fork()*..

```
int main() {  
    if (fork() == 0) {  
        exit(0)  
    } else {  
        fork()  
        fork()  
    }  
}
```

Solução que inverte o if para testar se estamos no processo pai..

```
int main() {  
    if (fork() > 0) {  
        fork()  
        fork()  
    }  
}
```