Instituto Superior de Engenharia do Porto
Departamento de Engenharia Informática

SISTEMAS OPERATIVOS I

Ficha 8

Versão 1.0

Maio de 2006

Luis Lino Ferreira
Maria João Viamonte
Berta Batista
Nuno Malheiro

Sugestões e participações de erros para:
llf@dei.isep.ipp.pt

1 – Comece por testar os exemplos da Figura 5, 6 e 7.

a) **No caso do programa da Figura 5 qual é a ordem com os processos são executados. Explique porquê.**

b) **Introduza a instrução `sleep(3)` alternadamente no pai e no filho. O que acontece?**

c) **Modifique o programa da Figura 5 de modo a que sejam gerados 2 filhos, do mesmo pai. Nota os filhos não devem gerar outros filhos.**

d) **Acrescente o código para que cada um dos processos da alínea anterior imprima o PID.**

e) **Modifique o programa da Figura 7 de modo a que sejam gerados 5 filhos. Cada um dos filhos simula a sua operação através da instrução `sleep(i)`, em que `i` é o número de ordem do processo. O processo pai deve esperar pelos cinco filhos, começando pelo filho5.**

2 – Faça um programa que crie um processo e:

- **O processo *pai* escreve os números de 1 a 10.**
- **O processo *filho* escreve os números de 11 a 20.**

Nota: Primeiro devem aparecer os números de 1 a 10, mas os dois processos devem correr paralelamente.

3 – Considere o programa seguinte:

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

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

 for (f = 0; f < 3; f++)
 {
     pid = fork(); /* Cria um PROCESSO */
     if (pid > 0) /* Código do PAI */
     {
          printf("Pai: Eu sou o PAI\n");
     }
     else /* Código do FILHO */
     {
          sleep(1);
     }
 }
} /* fim main */
```

a) **Analise o código deste programa e verifique quantos processo vão ser criados.**

b) **Teste o programa de modo a verificar se o resultado da alínea anterior está correcto.**

c) **Desenhe uma árvore que descreva os processos criados.**

d) **Altere o programa de modo a que apenas sejam gerados 3 filhos.**

e) **Altere o programa de modo a que todas as possíveis situações de erro sejam tratadas.**

f) **Altere o programa de modo que o pai espere pelo fim de cada um dos filhos.**

g) **Altere o programa de modo a que cada filho devolva o seu número de ordem ao pai. O pai deverá imprimir qual o número de ordem de cada um dos filhos que vai terminando, assim como o seu PID.**

h) **Altere o programa de modo a que o pai apenas espere pelo segundo filho, mas sem bloquear.**

4 – Considere o programa seguinte:

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

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

 fork();
 printf("1\n");
 fork();
 printf("2\n");
 fork();
 printf("3\n");

}
```

a) Desenhe uma árvore que descreva o conjunto dos processos criados.

b) Será possível que o número 1 apareça depois do 3? Porquê?

5 - Faça um programa que crie 2 processos e:

- Escreve "Eu sou o *pai*" no processo *pai*.
- Escreve "Eu sou o 1º *filho*" no primeiro *filho*.
- Escreve "Eu sou o 2º *filho*" no segundo *filho*.

6 - Faça um programa que crie um processo e:

- O processo *filho* escreve os números 1 .. 500.
- O *filho* quando terminar, "retorna" o valor 5.
- O processo *pai* escreve os números 501 .. 1000.
- O *pai* só deve terminar quando o *filho* terminar e tem de escrever o "valor de saída" do *filho*.

a) O que observa de estranho?

b) Modifique o seu programa de modo a garantir que os números apareçam de forma correcta.

7 - Faça um programa que crie 5 processos e:

- **Cada processo escreve 200 números:**
- **1º processo: 1 .. 200**
- **2º processo: 201 .. 400**
- **3º processo: 401 .. 600**
- **4º processo: 601 .. 800**
- **5º processo: 801 .. 1000**
- **O processo *pai* tem de esperar que todos os processos *filho* terminem.**

8 - Tendo um array de 1000 posições, faça um programa que crie 5 processos e:

- **Dado um número, procurar esse número no array.**
- **Cada processo *filho*, procura 200 posições.**
- **O processo que encontrar o número, deve imprimir a posição do array onde se encontra. Também deve "retornar" como valor de saída o número do processo (1, 2, 3, 4, 5).**
- **Os processos que não encontrarem o número devem "retornar" como *valor de saída* o valor 0.**
- **O processo *pai* tem de esperar que todos os *filhos* terminem e imprimir o número do processo onde esse número foi encontrado (1, 2, 3, 4, 5).**

Nota: O array não tem números repetidos.