

Processos

Aula 2 - Criação de processos

José Pedro Oliveira
(jpo@di.uminho.pt)

Grupo de Sistemas Distribuídos
Departamento de Informática
Escola de Engenharia
Universidade do Minho

Sistemas Operativos I
2006-2007



Conteúdo

- 1 Criação de processos concorrentes
- 2 Independência de processos
- 3 Divisão/delegação de tarefas
- 4 Referências



José Pedro Oliveira

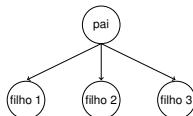
Processos

Criação de processos concorrentes

Criação de processos concorrentes

Exercício

Criar três processos concorrentes que tenham o mesmo progenitor. Pretende-se ainda que a implementação seja facilmente extensível.



José Pedro Oliveira

Processos

José Pedro Oliveira

Processos

Criação de processos concorrentes

Criação de processos concorrentes

Primeira abordagem

```

1 #include <unistd.h>
2
3 int main(void)
4 {
5
6     fork();
7     fork();
8     fork();
9
10    return 0;
11 }
  
```



José Pedro Oliveira

Processos

Criação de processos concorrentes

Segunda abordagem

```

1 #include <stdio.h>
2 #include <unistd.h>
3
4 int main(void)
5 {
6     int i;
7     for (i=0; i<3; i++) {
8         if (fork() == 0) {
9             printf("Filho\n");
10        }
11    }
12    return 0;
13 }

```



Conteúdo

- 1 Criação de processos concorrentes
- 2 Independência de processos
- 3 Divisão/delegação de tarefas
- 4 Referências



Criação de processos concorrentes

Terceira abordagem

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4
5 int main(void)
6 {
7     int i;
8     for (i=0; i<3; i++) {
9         if (fork() == 0) {
10            printf("Filho\n");
11            exit(0);
12        }
13    }
14    return 0;
15 }

```



Processos

Variável local

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/types.h>          /* pid_t */
5
6 int main(void)
7 {
8     int v = 5;
9
10    if (fork() == 0) { v++; }
11
12    printf("v = %d\n", v);
13
14    return 0;
15 }

```



Conteúdo

- 1 Criação de processos concorrentes
- 2 Independência de processos
- 3 Divisão/delegação de tarefas
- 4 Referências



Chamada ao sistema: wait

Extracto de código (processo pai)

```

1 q = wait(&status);
2
3 if (q == -1) {
4     /* Erro */
5 } else if (q > 0) {
6     /* q -> pid do processo que terminou */
7
8     if (WIFEXITED(status)) {
9         /* Processo q terminou normalmente */
10        /* Código de saída = WEXITSTATUS(status) */
11    } else {
12        /* Processo q terminou anormalmente! */
13    }
14 }

```



Divisão de tarefas

Considere o seguinte vector:

```
int v[] = { 0, 1, 0, 1, 0, 0, 1, 1 };
```

Exercício 1

O processo filho deve enviar ao processo pai o número de ocorrências do dígito 1 existentes no vector.

Exercício 2

O processo filho deve enviar ao processo pai o número de ocorrências do dígito 1 existentes em metade do vector. O processo pai deve determinar o número de ocorrências na outra metade do vector e esperar pelo resultado do processo filho, de modo a poder apresentar o resultado final.



Divisão de tarefas

Exercício 1

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/types.h>
5 #include <sys/wait.h>
6
7 int main(void)
8 {
9     pid_t p, q;
10    int i, status, count;
11    int v[] = { 0, 1, 0, 1, 0, 0, 1, 1 };
12
13    p = fork();
14    if (p == 0) {
15        count = 0;
16        for (i = 0; i < sizeof(v) / sizeof(v[0]); i++)
17            if (v[i] == 1) count++;
18        exit(count);
19    } else if (p > 0) {
20        q = wait(&status);
21        if (q > 0 && WIFEXITED(status)) {
22            printf("Contador: %d\n", WEXITSTATUS(status));
23        }
24    }
25    return 0;
26 }

```



Delegação de tarefas

Exercício

Utilizar vários processos filhos (workers) para verificar a existência de um dado número num vector.

Assumir:

- um vector de mil inteiros
- o número inteiro a procurar
- quatro processos trabalhadores (filhos)

Exercício 2

Determinar o número de ocorrências do número no vector.

Exercício 3

Determinar a posição absoluta da primeira ocorrência do número no vector.



Conteúdo

- 1 Criação de processos concorrentes
- 2 Independência de processos
- 3 Divisão/delegação de tarefas
- 4 Referências



Delegação de tarefas

Dicas para a resolução

- 1 o processo pai deverá criar os processos trabalhadores
- 2 cada processo filho deverá manipular um segmento do vector
- 3 cada processo filho deverá retornar o valor 1 ou 0 ao processo pai caso encontre, ou não, o valor a procurar no seu segmento do vector
- 4 o processo pai deverá recolher os valores retornados por todos os processos filhos e concluir se o número inteiro existe, ou não, no vector



Referências

Bibliografia

- **Advanced Programming in the UNIX Environment, 2nd ed.**
W. Richard Steven, Stephen A. Rago
<http://www.apuebook.com/>
 - Capítulo 1 - UNIX System Overview
 - Capítulo 7 - Process Environment
 - Capítulo 8 - Process Control
- **Linux Programming by Example: The Fundamentals**
Arnold Robbins
<http://authors.phptr.com/robbins/>
- **The Design of the Unix Operating System**
Maurice J. Bach
<http://www.phptr.com/title/0132017997>

