Pontifícia Universidade Católica do Rio de Janeiro

Lab 1 - fork() exec()

Sistemas Operacionais

Alunos	Tharik Lourenço
	Suemy Inagaki
Professor	Luiz Fernando Seibel

Rio de Janeiro, 22 de março de 2021

Conteúdo

1 Problema 1 **1**
1.1 Código Fonte 1
1.2 Resultado da Execução 2
1.3 Perguntas 4

2 Problema 2 **4**
2.1 Código Fonte 5
2.2 Resultado da Execução 6
2.3 Perguntas 7

3 Problema 3 **7**
3.1 Código Fonte 7
3.1.1 Programa Principal 7
3.1.2 Programas Auxiliares 8
3.2 Resultado da Execução 9
3.3 Perguntas 9

1 Problema 1

1) Faça um programa em que três processos executam em paralelo as seguintes ações:

- Pai - Imprime a soma dos números de 0 a 999. Antes de imprimir a soma, imprime a frase "Pai foi criado". Após imprimir o resultado da soma, imprime a frase "Processo pai vai finalizar" e finaliza quando o filho terminar.
- Filho - Imprime a soma dos números de 1000 a 1999. Antes de imprimir a soma, imprime a frase "Filho foi criado". Após imprimir a soma, imprime a frase "Processo filho vai finalizar" e finaliza quando o neto terminar.
- Neto - filho do processo Filho (ou seja, neto do processo Pai). Imprime a soma dos números de 2000 a 2999. Antes de imprimir a soma, imprime a frase "Neto foi criado". Após imprimir a soma, imprime a frase "Processo neto vai finalizar" e finaliza o processo.

1.1 Código Fonte

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

int main(void)
{
    char* const param[]={"/ bin / ls", "-l", "/ u / userid / dirname",NULL};
    int status;

    while (1) { /* repeat forever */
        int sum1 =0;
        int sum2 =0;
        int sum3 =0;
        int i =0;
        int j =1000;
        int k =2000;
        if (fork() == 0) {
            if(fork() == 0){ /* é o processo neto */
```

```
                printf("Neto foi criado\n");
                for (k=2000;k<=2999;k++){
                    sum3 +=k;
                    printf("neto (%d)\n", k);
                }
            printf("soma neto (%d)\n", sum3);
            printf("processo neto vai terminar\n");
            exit(0);
            }
            else { /* é o processo filho */
                printf("Filho foi criado\n");
                for (j=1000;j<=1999;j++){
                    sum2 +=j;
                    printf("filho (%d)\n", j);
                }
                printf("soma filho (%d)\n", sum2);
                printf("processo filho vai terminar\n");
                wait(&status); /* wait for child to exit */
                exit(0);
            }
        }
        else{ /*é o processo pai*/
                printf("Pai foi criado\n");
            for (i=0;i<=999;i++){
                sum1 +=i;
                printf("pai (%d)\n", i);
            }
            printf("soma pai (%d)\n", sum1);
            printf("processo pai vai terminar\n");
            wait(&status); /* wait for child to exit */
            exit(0);
        }
    }
         return 0;
}
```

1.2 Resultado da Execução

```
$ gcc lab1.c -o lab1
$ ./lab1

Pai foi criado
```

```
pai (0)
pai (1)
pai (2)
pai (3)
...
pai (825)
pai (826)
pai (827)
pai (82Filho foi criado
filho (1000)
filho (1001)
filho (1002)
...
filho (1310)
filho (1311)
filho (1312)
filho (131Neto foi criado
neto (2000)
neto (2001)
neto (2002)
neto (2003)
...
neto (2337)
neto (2338)
neto (2339)
3)
filho (1314)
filho (1315)
filho (1316)
...
filho (1626)
filho (1627)
filho (1628neto (2340)
neto (2341)
neto (2342)
neto (2343)
...
neto (2679)
neto (2680)
neto)
filho (1629)
filho (1630)
filho (1631)
...
filho (1941)
filho (1942)
filho (1943) (2681)
neto (2682)
neto (2683)
...
neto (2997)
neto (2998)
neto (2999)
soma neto (2499500)
processo neto vai terminar

filho (1944)
filho (1945)
filho (1946)
...
filho (1998)
filho (1999)
```

```
soma filho (1499500)
processo filho vai terminar
8)
pai (829)
pai (830)
pai (831)
...
pai (997)
pai (998)
pai (999)
soma pai (499500)
processo pai vai terminar
```

1.3 Perguntas

1. **É possível observar os processos executando em paralelo? Como?**

   Sim, é possivel observar os processos executando em paralelo. No início só o pai vai ser executado. Quando der o fork do filho, vai executar o pai e o filho. Depois quando der fork no neto, vai executar o pai, o filho e o neto. Os três vão estar executando, cada hora imprimindo a mensagem de um deles, como podemos ver no resultado acima.

   Quando o processo pai somou o valor 828, o processo filho foi iniciado e quando o filho somou o valor 1313, o processo neto foi criado. E ficaram alternando sucessivamente até que o processo neto terminou, depois o filho terminou e o pai retomou a execução de onde parou, até finalizar também.

2. **Houve concorrência entre os processos que afetou o resultado da soma?**

   A concorrência pode ser observada quando os valores são exibidos de forma misturada, mas o resultado da soma não foi afetado.

Dica: Para cada processo, dentro dos loops, exiba os resultados intermediários do valor a ser somado, o valor da soma, o pid do processo e o pid do processo pai.

2 Problema 2

Faça o mesmo programa do item 1 porém com os sub processos em um mesmo nível.

2.1 Código Fonte

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

int main()
{
    int status;

    int sum1 =0;
    int sum2 =0;
    int sum3 =0;
    int i =0;
    int j =1000;
    int k =2000;
    if (fork() == 0) { /* child 1*/
        printf("processo 1 foi criado\n");
        for (i;i<=999;i++){
            sum1 +=i;
            printf("processo 1 (%d)\n", i);
        }
        printf("soma processo 1 (%d)\n", sum1);
        printf("processo 1 vai terminar\n");
        printf("pid proc1 = %d, pid do pai = %d\n", getpid(), getppid());
        exit(0);
    }
    else if(fork() == 0) {  /* child 2*/
        printf("processo 2 foi criado\n");
        for (j;j<=1999;j++){
            sum2 +=j;
            printf("processo 2 (%d)\n", j);
        }
        printf("soma processo 2 (%d)\n", sum2);
        printf("processo 2 vai terminar\n");
        printf("pid proc2 = %d, pid do pai = %d\n", getpid(), getppid());
        exit(0);
    }
    else if (fork() == 0) { /* child 3*/
```

```
        printf("processo 3 foi criado\n");
        for (k;k<=2999;k++){
            sum3 +=k;
            printf("processo 3 (%d)\n", k);
        }
        printf("soma processo 3 (%d)\n", sum3);
        printf("processo 3 vai terminar\n");
        printf("pid proc3 = %d, pid do pai = %d\n", getpid(), getppid());
        exit(0);
    }
    else if(fork() > 0){
        printf("processo pai\n");
        printf("pid do processo pai = %d\n", getpid());
    }
    return 0;
}
```

2.2 Resultado da Execução

```
processo 1 foi criado
processo 1 (0)
processo 1 (1)
processo 1 (2)
...
processo 1 (997)
processo 1 (998)
processo 1 (999)
soma processo 1 (499500)
processo 1 vai terminar
pid proc1 = 79, pid do pai = 78
processo 2 foi criado
processo 2 (1000)
processo 2 (1001)
processo 2 (1002)
...
processo 2 (1997)
processo 2 (1998)
processo 2 (1999)
soma processo 2 (1499500)
processo 2 vai terminar
pid proc2 = 80, pid do pai = 78
processo 3 foi criado
processo 3 (2000)
processo 3 (2001)
processo 3 (2002)
...
processo 3 (2997)
processo 3 (2998)
processo 3 (2999)
soma processo 3 (2499500)
processo 3 vai terminar
pid proc3 = 81, pid do pai = 78
```

```
processo pai
pid do processo pai = 78
```

2.3 Perguntas

1. **É possível observar os processos executando em paralelo? Como?**

   Não é possível observar os processos executando em paralelo, pois o processo 2 só é executando quando o processo 1 acaba e o processo 3 só é executando quando o processo 2 acaba:

   proc1 $\Longrightarrow$ proc2 $\Longrightarrow$ proc3

2. **Houve concorrência entre os processos que afetou o resultado da soma?**

   Não houve concorrência entre os processos e o resultado da soma não foi afetado.

3 Problema 3

Implemente os seguintes programas:

- O primeiro exibe uma mensagem, por exemplo: "bom dia!", de nome bomdia
- O segundo deve ter a mesma funcionalidade do programa echo do Unix, que exibe no terminal os argumentos do programa, de nome meuecho

  ex: echo bom dia a resposta do programa é: >bom dia

Agora elabore um programa que utilize a estrutura de forks do item anterior e execute três programas: os que você escreveu (bomdia e meuecho) e o utilitário do sistema (ex:"cat <nome de arquivo>").

Utilize alguma função da família "exec"para realizar esta atividade.

***DICA**: Para saber os protótipos das funções disponíveis para a execução de programas, execute o comando "man" no Terminal ("man exec", "man execv", ...).*

3.1 Código Fonte

3.1.1 Programa Principal

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

```c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char* argv[])
{
    int status;
    char *LocalE[]={"./echo,", argv[1], NULL};
    char *LocalB[]={"./bomdia",NULL};
    while (1) { /* repeat forever */
        if (fork() == 0) {
            if(fork() == 0) { /*processo neto*/
                printf("neto foi criado\n");
                execve("./echo", LocalE, NULL); /* execute command */
                printf("processo neto vai terminar\n");
                exit(0);
            }
            else{ /* processo filho*/
                printf("filho foi criado\n");
                execve("./bomdia", LocalB, NULL); /* execute command */
                printf("processo Filho vai terminar\n");
                wait(&status); /* wait for child to exit */
                exit(0);
            }
        }
        else{
            printf("pai foi criado\n");
            execl("/bin/sh", "sh", "-c", "/bin/cat algo.txt", NULL);
            printf("processo pai vai terminar\n");
            wait(&status); /* wait for child to exit */
            exit(0);
        }

    }
    return 0;
}
```

3.1.2 Programas Auxiliares

```c
/* bomdia.c */
#include <stdio.h>
```

```c
void exibe(){
    printf("bom dia!\n");
}
int main(){
    exibe();
    return 0;
}
```

```c
/* meuecho.c */
#include <stdio.h>

int main(int argc, char* argv[]){
    if(argc > 1)
        for(int i = 1; i < argc; i++){
            if(i < argc -1)
                printf("%s ", argv[i]);
            else
                printf("%s\n", argv[i]);
        }
    return 0;
}
```

algo.txt

usando o execl para chamar o cat

3.2 Resultado da Execução

```
$ gcc lab1-3.c -o lab13
$ ./lab13 "testando o programa meuecho"

pai foi criado
filho foi criado
neto foi criado
testando o programa meuecho
bom dia!
usando o execl para chamar o cat
```

3.3 Perguntas

1. **O que você observou em termos de semelhanças e diferenças para executar os programas? Indique como você decidiu implementar os programas (estrutura hierárquica ou no mesmo nível) e justifique sua decisão.**

Os programas bomdia e meuecho conseguimos executar com o execve, mas não conseguimos usar o mesmo para o utilitário cat. Para o cat tivemos que chamar o execl e passar como argumentos o path para a shell e depois chamar o cat com o parametro algo.txt.

Decidimos implementar o programa com estrutura hierárquica, pois acreditamos ser melhor para a utlização do exec. Como o mesmo não permite que o programa seja retomado de onde parou e "mata"o processo onde se encontra, não faria sentido não associar sua chamada a um fork().