



Primeira Lista de Exercícios de Sistemas Operacionais

01) Implemente um programa C que recebe como parâmetros na linha de comando 10 números inteiros (desordenados). O programa MAIN armazena os números em um array e cria um processo filho. Em seguida o MAIN deve ordenar o array usando “ordenação simples” enquanto o filho deve fazer “quick sort”. Ao final da ordenação, cada processo deve exibir o tempo gasto para realizar a mesma. O Pai deve fazer um wait() para esperar o filho terminar.

Dicas:

```
#include <time.h>
...
clock_t c2, c1; /* variáveis que contam ciclos do processador */
float tmp;
c1 = clock();
//... código a ser executado
c2 = clock();
tmp = (c2 - c1)*1000/CLOCKS_PER_SEC; // tempo de execução em milissegundos
-----
void quickSort(int valor[], int esquerda, int direita){

    int i, j, x, y;

    i = esquerda;
    j = direita;
    x = valor[(esquerda + direita) / 2];
    while(i <= j){
        while(valor[i] < x && i < direita){
            i++;
        }
        while(valor[j] > x && j > esquerda){
            j--;
        }
        if(i <= j){
            y = valor[i];
            valor[i] = valor[j];
            valor[j] = y;
            i++;
            j--;
        }
    }
}
```

```

    }
}
if(j > esquerda){
    quickSort(valor, esquerda, j);
}
if(i < direita){
    quickSort(valor, i, direita);
}
}

```

02) Analise o trecho de código a seguir e determine quantos processos são criados. Assuma que nenhum erro ocorre ao realizar a chamada *fork()*. Desenhe uma árvore que mostre como os processos estão relacionados. Nessa árvore, cada processo será representado por um nó (um círculo contendo um número que representa em qual chamada *fork()* ele foi criado). O processo original deve conter '0', o processo criado no 1º. *fork()* deve conter 'F1', o processo criado no 2º. *fork()* deve conter 'F2' e assim sucessivamente. Deve haver setas (apontando p/ baixo) saindo de cada pai em direção a cada filho.

```

c2 = 0;
c1 = fork(); /* fork 1 */
if (c1 == 0)
    c2 = fork(); /* fork 2 */
fork(); /* fork 3 */
if (c2 > 0)
    fork(); /* fork 4 */

```

03) Um grafo de precedência é um grafo direcionado em que a relação $(a) \rightarrow (b)$ indica que 'a' precede 'b'. Neste exercício, os nós do grafo devem representar as instruções numeradas no código abaixo. Desta forma o grafo de precedência representará a ordem em que as instruções serão executadas. Observe que se o programa não tivesse *fork()*, o grafo seria linear pois a execução desse programa seria uma sequência de instruções (com desvios ou não). Ex:

(1) \rightarrow (2) \rightarrow (8) \rightarrow (9) \rightarrow (8) ...

No entanto sempre que há um *fork()*, passamos a ter execuções em paralelo. Ex:

\rightarrow (3) ...
 (1) \rightarrow (2) /
 \
 \rightarrow (3) \rightarrow 4

Desenhe o grafo de precedência referente ao código a seguir:

//processoPai.c

```
int f1, f2, f3; /* Identifica processos filho*/
int main(){
[1]   printf("Alo do pai\n");
[2]   f1 = fork;
[3]   if (f1==0)
[4]       execlp("codigo_filho","codigo_filho",NULL);
[5]   printf("Filho 1 criado\n");
[6]   f2 = fork;
[7]   if (f2==0)
[8]       execlp("codigo_filho","codigo_filho",NULL);
[9]   printf("Filho 2 criado\n");
[10]  waitpid(f1,null,0);
[11]  printf("Filho 1 morreu\n");
[12]  f3 = fork;
[13]  if (f3==0)
[14]      execlp("codigo_filho","codigo_filho",NULL);
[15]  printf("Filho 3 criado\n");
[16]  waitpid(f3,null,0);
[17]  printf("Filho 3 morreu\n");
[18]  waitpid( f2,null,0);
[19]  printf("Filho 2 morreu\n");
[20]  exit();
}
```

//processoFilho.c

```
int main() {
[21]   printf("Alo do filho\n");
[22]   exit();
}
```

04) Escreva um programa, em Java, que funcione como um shell simples. O programa deve receber, pela linha de comando, o comando do usuário (ex. "cat file.txt") e depois criar um processo externo separado que executa este comando.

05) Implemente um programa C que busque em um array desordenado um dado elemento. O tamanho do array deve ser informado via linha de comando (máximo de 128 posições) e seus elementos preenchidos de forma aleatória, sem repetições (valores de 0 até tamanho -1). A busca de ser realizada por dois processos filhos. O primeiro irá realizar a busca do início até o final do array, enquanto o segundo irá realizar a busca do final até o início. O primeiro processo filho deve esperar uma quantidade aleatória de tempo entre 0 e 1000 microssegundos. O programa pai espera até que o primeiro dos processos filhos finalize a busca. O status de retorno do processo filho

(informado pela função `exit`) determina o valor do índice onde o elemento foi encontrado (-1 indica que o elemento não foi encontrado).

Dicas:

- a função `wait(status_ptr)` pode ser usada para esperar por qualquer um dos processos filhos finalizar;
- a função `WEXITSTATUS(*status_ptr)` retorna os 8 bits de menor ordem do sinal de retorno. Para obter o valor retornado por um processo filho, utilize uma variável do tipo inteiro de 1 byte (`int8_t`). Este foi definido na biblioteca `<stdint.h>`;
- inicialize a semente usada para gerar números aleatórios com a hora atual (função `srand(int seed)`), de modo que a sequência de números aleatórios gerados não seja determinística. Utilize a biblioteca `<time.h>`;

06) Modifique o exercício 04 de modo a introduzir o operador de pipeline '|'. Este operador é utilizado para concatenar processos, tal que a saída de um processo seja redirecionada para a entrada do próximo processo. Exemplos de utilização deste operador incluem:

```
> ls -l | grep .java  
> who | cut -c 1-16,26-42  
> ls | grep .java | grep Example
```

Dica:

- Considere o uso do método estático `startPipeline`;

07) A conjectura de Collatz preocupa-se com o que acontece quando tomamos um inteiro positivo N qualquer e aplicamos o seguinte algoritmo:

$$N = \begin{cases} N/2, & \text{se } N \text{ é par} \\ 3 \times N + 1, & \text{se } N \text{ é ímpar} \end{cases}$$

A conjectura estabelece que, quando esse algoritmo é aplicado continuamente, todos os inteiros positivos acabam chegando a 1. Por exemplo, se $N = 35$, a sequência é: 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1. Escreva um programa em C que gere essa sequência no processo filho. O número inicial será fornecido na linha de comando. Por exemplo, se o número 8 for passado como parâmetro na linha de comando, o processo-filho exibirá 8, 4, 2, 1. Já que os processos pai e filho

têm suas próprias cópias dos dados, será necessário que o filho exiba a sequência. O processo pai deve esperar que o processo filho conclua antes de encerrar. Execute a verificação de erros necessária para garantir que um inteiro positivo seja passado na linha de comando.