Relatório EP 1 · SO

Prof. Dra. Gisele da Silva Craveiro

Mikael Gi Sung Shin, 10843441

Exercício 1:

Código (comentários para cada linha):

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

int main() // processo pai ou processo chamador
{

    pid_t childProcess;     // criando um id de processo para a variável
    childProcess = fork();  // criando o processo filho

    int i, status;          // atribuindo as variáveis para o código

    if(childProcess == 0){  // verifica se é processo filho, e executa o 'if' caso seja.

        printf("Hello World! I'm child process.\n"); // print de Hello World do processo filho

        _exit(0); // fim do processo filho, passando o controle para o processo pai

    } else { // executa o 'else' referente ao processo pai

        wait(childProcess, &status, 0); // suspende a execução do processo pai até que seja finalizado
        printf("Hello World! I'm parent process.\n"); // print de Hello World do processo pai
    }

    return 0;
}

```

Saída:

```
Hello World! I'm child process.
Hello World! I'm parent process.


...Program finished with exit code 0
Press ENTER to exit console.
```

Explicação do código:

Primeiramente, é necessário importar as devidas bibliotecas de C, para que a criação e finalização dos processos sejam executados. As linhas 1 e 4, importam estas bibliotecas, além de outras padrão que não são específicas para a “manipulação” de processos.

Para este exercício, tive de declarar um ID do processo filho, cujo o nome é “childProcess”. Em seguida, esta variável recebe a função fork() a fim de criar definitivamente o processo filho que será uma

cópia do pai a partir do ponto em que este processo foi instanciado. No caso de sucesso de criação do processo filho, o PID (ID do processo) recebe 0, e -1, caso contrário. Dessa forma, foi verificado se realmente tal processo foi criado. Se sim, executa o "Olá Mundo! Sou o processo filho." e o finalizamos com a função _exit(). A função wait() suspende a execução processo pai/chamador (no caso, a main) até que ocorra a finalização do filho. Finalizado, as linhas de código do processo pai será executado ("Olá Mundo, sou o processo pai.").

Instruções para compilação:

1) Usar alguma IDE que suporte a linguagem C, e execute ("run") o código descrito acima. (obs: utilizei o compilador online https://www.onlinegdb.com/online_c_compiler, por apresentar erros no CodeBlocks dentro do Windows).
2) Executar as seguintes linhas de código em algum terminal, dentro do diretório do programa:
```
gcc "nomedoarquivo".c -o "nomedoarquivo"
./"nomedoarquivo"
```

Referências para o desenvolvimento do código:

1) https://www.tutorialspoint.com/process-vs-parent-process-vs-child-process#:~:text=A%20child%20process%20is%20a,inherits%20most%20of%20its%20attributes.
2) https://www.dca.ufrn.br/~adelardo/cursos/DCA409/node36.html
3) https://www.youtube.com/watch?v=3VgLkCqqKWo&t=388s&ab_channel=BrunoSampaioPinhodaSilva

---

Exercício 2:

Código (comentários para cada linha):

```
public class Ex2 {

    public static void main(String[] args) {

        for(int i = 1; i <= 20; i++){ // loop para instânciar as threads

            String[] name = new String[21]; // alocando 21 posições para o nome [0] - [20]
            int[] time = new int[21]; // alocando 21 posições para o tempo de sleep [0] - [20]

            name[i] = "Thread# " + i; // atribuindo nome à thread (Thread #n, sendo n o número do contador)
            time[i] = 1000 * i; // atribuindo tempo de sleep para a thread (para Thread #3, o tempo será de 1s * 3)

            if(i <= 10 || i == 20) { // instanciando a Thread #1 à #10 e pulando direto para a Thread #20.

                MyThreads[] thread = new MyThreads[21]; // declarando o obj. thread
                thread[i] = new MyThreads(name[i], time[i]); // instanciando o obj. passando os parâmetros esperados
            }
        }

    }

}
```

```java
public class MyThreads extends Thread { // classe MyThreads herdando a classe "Thread" do Java

    public String name; // atributo nome da thread
    public int time; // atributo tempo de sleep da thread

    public MyThreads(String name, int time){ // construtor

        this.name = name;
        this.time = time;
        start(); // método que está dentro da classe Thread
                 // serve para tornar a Thread pronta para ser executada
    }

    public void run(){ // método implementado para executar as threads instanciadas pela main (class Ex2)

        int showTime = 0; // variável que mostra todos os tempos de execução de todas as threads

        try {
            for (int i = 0; i <= 10; i++){ // loop para a execução de threads

                System.out.println(this.name + "\t Tempo: " + showTime/1000 + "s\t"); // output do programa
                Thread.sleep(this.time); // tempo que o thread irá dormir
                showTime += this.time; // (somador) recebe o tempo de sleep da "Thread #n" + ela própria

            }
        } catch(Exception e) { // tratamento da exceção, caso ocorra alguma
            e.printStackTrace(); // print do tipo de exceção
        }
    }
}
```

Saída:

```
Thread# 2       Tempo: 0s
Thread# 1       Tempo: 0s
Thread# 3       Tempo: 0s
Thread# 4       Tempo: 0s
Thread# 5       Tempo: 0s
Thread# 6       Tempo: 0s
Thread# 7       Tempo: 0s
Thread# 8       Tempo: 0s
Thread# 9       Tempo: 0s
Thread# 10      Tempo: 0s
Thread# 20      Tempo: 0s
```

(1)

```
Thread# 1       Tempo: 1s
Thread# 1       Tempo: 2s
Thread# 2       Tempo: 2s
Thread# 3       Tempo: 3s
Thread# 1       Tempo: 3s
Thread# 4       Tempo: 4s
Thread# 2       Tempo: 4s
Thread# 1       Tempo: 4s
```

(2)

```
Thread# 20      Tempo: 100s
Thread# 10      Tempo: 100s
Thread# 20      Tempo: 120s
Thread# 20      Tempo: 140s
Thread# 20      Tempo: 160s
Thread# 20      Tempo: 180s
Thread# 20      Tempo: 200s
```

(3)

Explicação do código:

Para este exercício, diferentemente do primeiro, procurei por imprimir o nome das threads criadas e o tempo de execução de cada um delas, ao invés dos “Hello World’s”, pois, desse modo, facilita a identificação do que cada thread está executando. Foi preciso, então, instanciar várias threads (no total de 11, isso foi feito no método main) passando para o construtor da classe “MyThreads”, o seu nome (Thread #n) e seu tempo de sleep (“dormida”, o número do thread vezes 1 segundo). Após a instanciação, a thread está pronta para ser executada devido ao método start(), originário da classe Thread do próprio Java, que está no próprio construtor. Em seguida, as threads são executadas pelo método run(), que irá imprimir na tela, o seu nome e o tempo que tal instrução foi executada (em segundos). Essa tarefa, se repete por 10 vezes e cada um dos threads terá o tempo de sleep para que seja executada a próxima instrução. Pelo print (1) acima, enxerga-se que todas as instâncias começam no 0s. Em 1s, a Thread #1 é executada, em 2s, a Thread #1 e #2 são executadas, em 3s, a Thread #1 e #3, em 4s, a Thread #1, #2 e #4, como visto no print (2), portanto, a Thread #10 será executada de 10 a 10 segundos, e a Thread #20, de 20 a 20. No print (3), demonstra que a Thread #20 teve as últimas impressões, enquanto todas as outras threads finalizaram sua execução, por justamente ter um tempo de sleep maior que os demais.

Instruções para compilação:

1) Usar alguma IDE que suporte a linguagem Java, e execute (“run”) o código descrito acima.
2) Executar as seguintes linhas de código em algum terminal, dentro do diretório do programa:
   `javac “nomedoarquivo”.java`
   `java “nomedoarquivo”`

Referências para o desenvolvimento do código:

1) https://www.youtube.com/watch?v=v5l30QMKv6c&ab_channel=LoianeGroner
2) https://www.youtube.com/watch?v=rVsyXGwbmag&t=27s&ab_channel=MichelliBrito

---

Exercício 3:

Código (comentários para cada linha):

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

#define tam 10

void wait(){ // função que será executada, para simular um wait nos threads

        int i;
        int num = 100000000;
        for(i = 0; i < num; i++){
                num = num;
        }
}

void *run (void *arg){ // função que irá servir de execução para cada uma das threads

        int *valor; // declarando um ponteiro de inteiro
        valor = arg; // "valor" recebe o conteúdo do endereço de "arg"

        printf("Thread #%i: Hello World! \n", *valor); //

        wait(); // chamada a função que simula uma espera das threads

        printf("Thread #%i: See you later World! \n", *valor);
}

int main(){

        pthread_t threads[tam]; // declarando as threads posix

        int i, arg[tam]; // declarando as variáveis inteiras

        for(i = 0; i < tam; i++){ // loop: contador de 0 a 9

                arg[i] = i + 1; // array de argumento, posição 0 armazena 1, posição 1 armazena 2, ...
                pthread_create(&threads[i], NULL, run, (void*)&arg[i]); // instanciação das threads (de 1 à 10)
                //pthread_create:  1º parametro: recebe o endereço da thread
                //                 2º parametro: recebe atributos para a thread (nesse caso, usou-se o NULL)
                //                 3º parametro: recebe a função de início para a execução das threads
                //                 4º parametro: recebe o mesmo argumento que é passado para a função de início

        }


        for(i = 0; i < tam; i++){
                pthread_join(threads[i], NULL); // aguarda o término de cada uma das threads
                //pthread_join: 1º parametro: recebe a thread
                //              2º parametro: se !NULL, recebe o código de retorno da Thread (0 = sucesso, 'n' = código do erro)
        }

        printf("\nTodas as threads foram executadas.\n"); // mensagem de fim da execução das threads

        pthread_exit(NULL); // finalização das threads

        return 0;
}
```

Saída:

```
mikaelshin@mikaelshin-VirtualBox:~/Documentos$ gcc Ex3.c -o Ex3 -lpthread
mikaelshin@mikaelshin-VirtualBox:~/Documentos$ ./Ex3
Thread #6: Hello World!
Thread #7: Hello World!
Thread #5: Hello World!
Thread #4: Hello World!
Thread #8: Hello World!
Thread #3: Hello World!
Thread #9: Hello World!
Thread #10: Hello World!
Thread #2: Hello World!
Thread #1: Hello World!
Thread #1: See you later World!
Thread #2: See you later World!
Thread #5: See you later World!
Thread #9: See you later World!
Thread #3: See you later World!
Thread #8: See you later World!
Thread #10: See you later World!
Thread #6: See you later World!
Thread #7: See you later World!
Thread #4: See you later World!
```

Explicação do código:

Os desafios para o desenvolvimento do exercício 3, eram a compreensão dos métodos existentes na biblioteca <pthread.h> e como utilizá-los.

O “pthread_t” é o tipo que irá declarar as threads. O método “pthread_create(_, _, _, _)” é responsável por instanciar as threads. Para isso, deve-se passar os parâmetros necessários: o endereço da thread, atributos para thread (nesse caso, passamos nulo), função de início para a execução das threads (método run(), que irá imprimir a primeira mensagem, depois executa um método chamado wait(), simulando um estado de espera das threads, e depois imprime a segunda mensagem) e o argumento que é passado para a função de início. Com todos estes dados fornecidos, criamos as threads. O “pthread_join(_, _)” aguarda o término da execução da thread, para que, a partir daí, seja executada as próximas linhas de código. E por fim, as threads são finalizadas com o método pthread_exit(_).

Instruções para compilação:

1) Usar alguma IDE que suporte a linguagem C, e execute (“run”) o código descrito acima.
2) Executar as seguintes linhas de código em algum terminal, dentro do diretório do programa:
   `gcc “nomedoarquivo”.c -o “nomedoarquivo”`
   `./“nomedoarquivo”`

Referências para o desenvolvimento do código:

1) https://www.youtube.com/watch?v=uA8X5zNOGw8&ab_channel=JacobSorber
2) https://www.youtube.com/watch?v=CylpD8zXHZA&ab_channel=BrunoSampaioPinhodaSilva
3) https://www.youtube.com/watch?v=Z7BApQ9g4cI&t=488s&ab_channel=FelipeGraffa
4) https://www.youtube.com/watch?v=cwT3EJJQhlo&ab&ab_channel=DucaSiqueira