



Escola de Artes, Ciências e Humanidades
da Universidade de São Paulo

EDGAR HENRIQUE DE OLIVEIRA LIRA (12717266)
PEDRO AUGUSTO DE MELO DELAMURA SOARES (12542800)

SISTEMAS OPERACIONAIS

EP1

SÃO PAULO
2022

EDGAR HENRIQUE DE OLIVEIRA LIRA (12717266)
PEDRO AUGUSTO DE MELO DELAMURA SOARES (12542800)

SISTEMAS OPERACIONAIS

EP1

Trabalho apresentado à Escola de Artes,
Ciências e Humanidades da Universidade de
São Paulo como requisito da disciplina de
Sistemas Operacionais.

Prof. Dr.^a Gisele da Silva Craveiro.

SÃO PAULO
2022

SUMÁRIO

1 LINGUAGEM C - Fork	3
1.1 Código C do Fork	3
1.2 Output em C de Fork	3
1.3 Implementação, compilação e execução de fork em C	3
2 LINGUAGEM JAVA - Threads	3
2.1 Organização dos arquivos	3
2.2 Código em Java de Threads	4
2.3 Output Threads em Java	5
2.4 Implementação, compilação e execução de Threads em Java	5
3 LINGUAGEM C - Threads	7
3.1 Código em C de Threads	7
3.2 Output Threads em C	8
3.3 Implementação, compilação e execução de Threads em C	8

1 LINGUAGEM C - Fork

1.1 Código em C do fork:

```
#include <stdio.h> //Biblioteca que implementa a função usada printf
#include <unistd.h> //Biblioteca que implementa a função usada fork
```

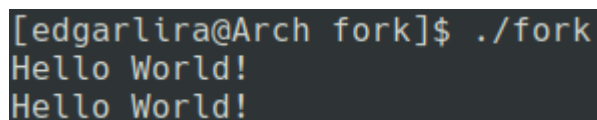
```
int main() { // Declaração da função Main
```

```
    fork(); // Função que cria um processo filho igual ao processo pai
    printf("Hello World!\n"); // Imprime Hello World
```

```
    return 0; // Indica que o programa acabou
```

```
}
```

1.2 Output em C de Fork:



```
[edgarlira@Arch fork]$ ./fork
Hello World!
Hello World!
```

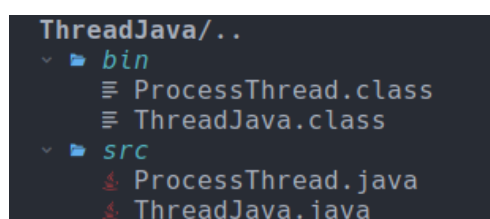
1.3 Implementação, compilação e execução de fork em C:

Para implementar um processo filho (fork) foi necessário adicionar uma biblioteca do C e utilizar a função fork antes de chamar a função printf, para assim ser iniciado um processo filho que executaria as mesmas instruções que o processo pai a partir daquele ponto. Na compilação foi utilizado o comando gcc usando a flag “lpthread” da seguinte maneira “gcc fork.c -o fork -lpthread” e a execução foi utilizando o binário gerado, portanto foi utilizado o seguinte comando no terminal “./fork”.

2. LINGUAGEM JAVA - Threads

2.1 Organização dos arquivos:

Com a finalidade de deixar o código mais limpo criei 2 classes, a ThreadJava.java contém o método Main e a classe ProcessThread.java é a classe da Thread.



```
ThreadJava/..
├── bin
│   ├── ProcessThread.class
│   └── ThreadJava.class
└── src
    ├── ProcessThread.java
    └── ThreadJava.java
```

2.2 Código em Java de Threads:

```
/* Arquivo -> ThreadJava.java */
```

```
import java.lang.Thread;
```

```
import java.lang.Math;
```

```
public class ThreadJava {
```

```
    public static void main(String[] args) throws Exception {
```

```
        // Instanciando os objetos para iniciar as Threads
```

```
        ProcessThread thread1 = new ProcessThread(0, (int) (Math.random() * 5000));
```

```
        ProcessThread thread2 = new ProcessThread(1, (int) (Math.random() * 5000));
```

```
        ProcessThread thread3 = new ProcessThread(2, (int) (Math.random() * 5000));
```

```
        // Iniciando as Threads
```

```
        thread1.start();
```

```
        thread2.start();
```

```
        thread3.start();
```

```
    }
```

```
}
```

```
/*                                Arquivo -> ProcessThread.java                                */
```

```
//criando a classe que estende a classe mãe Thread
```

```
public class ProcessThread extends Thread {
```

```
    //variáveis da classe
```

```
        int id;
```

```
        int tempo;
```

```
    //construtor da classe
```

```
    public ProcessThread(int id, int tempo) {
```

```
        this.id = id;
```

```
        this.tempo = tempo;
```

```
    }
```

```
/* O objetos que serão instanciados poderão usar esse método para assim  
   analisarmos as threads*/
```

```
    public void run() {
```

```

try {
//Printa o tempo em que o processo irá dormir
    System.out.println("ID = " + id + " vai dormir por " + tempo + ("ms"));
//processo dorme
    Thread.sleep(tempo);
//Após o tempo do “sleep” acabar o processo printa Hello World e seu ID
    System.out.println("Hello World, ID = " + id);
/* Catch é executado somente se o “try” retornar alguma exceção */
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
}
}

```

2.3 Output Threads em Java:

```

[edgarlira@Arch bin]$ java ThreadJava && java ThreadJava && java ThreadJava
ID = 1 vai dormir por 3914ms
ID = 0 vai dormir por 4169ms
ID = 2 vai dormir por 3135ms
Hello World, ID = 2
Hello World, ID = 1
Hello World, ID = 0
ID = 0 vai dormir por 4540ms
ID = 2 vai dormir por 490ms
ID = 1 vai dormir por 3624ms
Hello World, ID = 2
Hello World, ID = 1
Hello World, ID = 0
ID = 1 vai dormir por 3196ms
ID = 0 vai dormir por 512ms
ID = 2 vai dormir por 1903ms
Hello World, ID = 0
Hello World, ID = 2
Hello World, ID = 1

```

2.4 Implementação, compilação e execução de Threads em Java

Para implementar as Threads em java, foi necessário adicionar uma biblioteca com a classe Thread, criar uma classe que utilizava o conceito de herança de orientação a objetos, ou seja, a classe criada estende a classe Thread e nesta classe criada são implementados os métodos necessários para instanciar os objetos das threads e iniciar os métodos com suas instruções. Após criar a classe e instanciar os objetos da classe, basta executar o código e conseguiremos observar cada execução sendo feita paralelamente, como todas as threads executavam os

mesmos métodos, adicionei a função sleep e um gerador aleatório de números para definir quanto tempo um processo irá dormir, para assim observarmos mais calmamente a execução de cada thread.

Após a implementação estar completa, bastou compilar os arquivos, com o comando “javac ThreadJava.java” e a execução é feita com o binário gerado após a compilação, executando o ThreadJava.class, com o seguinte comando “java ThreadJava”.

3. LINGUAGEM C - Threads

3.1 Código Threads C:

```
#include <pthread.h> //Biblioteca para Threads
#include <stdio.h> //Biblioteca para o printf
#include <stdlib.h> //A função Sleep depende desta biblioteca para funcionar
#include <unistd.h> //Biblioteca da função de sleep
//função que será executada pelas Threads
void *functionThread(void *id) {
    //passa o id da Thread para variável ID
    long *ID = (long *)id;
    // Seleciona um valor aleatório entre 0 e 5 e atribui ele a variável tempo
    int tempo = rand() % 5;
    // Printa o valor de tempo em que a thread irá dormir
    printf("Thread %ld vai dormir por: %ds\n", *ID, tempo);
    //coloca a thread para dormir
    sleep(tempo);
    //printa HelloWorld e o ID da thread
    printf("HelloWorld, ID = %ld\n", *ID);
    //Retorno opcional (função do tipo void)
    return NULL;
}

int main(void) {
    //Declarando as variáveis
    pthread_t id0, id1, id2;
    //adiciono o ponteiro para memória delas em um array
    pthread_t *Theads[] = {&id0, &id1, &id2};
    // Cria um loop para criar as threads
    for (int k = 0; k < 3; k++)
        pthread_create(Theads[k], NULL, functionThread, (void *)Theads[k]);
    // função de Exit para as threads
    pthread_exit(NULL);
    // indica que a execução terminou
    return 0;
}
```


3.2 Output de Threads C:

```
[edgarlira@Arch ThreadC]$ ./ThreadC && ./ThreadC && ./ThreadC
Thread 139795528484544 vai dormir por: 3s
Thread 139795520091840 vai dormir por: 1s
Thread 139795511699136 vai dormir por: 2s
HelloWorld, ID = 139795520091840
HelloWorld, ID = 139795511699136
HelloWorld, ID = 139795528484544
Thread 139799063168704 vai dormir por: 3s
Thread 139799046383296 vai dormir por: 2s
Thread 139799054776000 vai dormir por: 1s
HelloWorld, ID = 139799054776000
HelloWorld, ID = 139799046383296
HelloWorld, ID = 139799063168704
Thread 140545875080896 vai dormir por: 3s
Thread 140545883473600 vai dormir por: 1s
Thread 140545866688192 vai dormir por: 2s
HelloWorld, ID = 140545883473600
HelloWorld, ID = 140545866688192
HelloWorld, ID = 140545875080896
```

3.3 Implementação, compilação e execução de threads em C:

Para implementar as Threads em C foi necessário adicionar as bibliotecas que continham as funções necessárias para iniciar as threads, após isso iniciei as variáveis das threads, além de utilizar os métodos que criavam cada thread, após isso foi necessário criar as funções as quais as threads executam e no final chamar uma função para dar fim a execução das threads(`pthread_exit`).

A compilação foi feita com o gcc, passando como flag “`lpthread`”, para o compilador ter certas diretrizes a seguir, mais especificamente usando o comando “`gcc ThreadC.c -o ThreadC -lpthread`” e a execução foi feita com o binário gerado a partir da compilação com o comando “`./ThreadC`”.