



Universidade do Estado do Rio de Janeiro – UERJ

Campus Regional Instituto Politécnico do Estado do Rio de Janeiro - IPRJ

Curso de Graduação em Engenharia da Computação

TRABALHO DE SISTEMAS OPERACIONAIS

Leonardo Simões

Professor:

Guilherme de Melo Baptista Domingues

Nova Friburgo, 17 de Janeiro de 2019.

SUMÁRIO

	Página
1. OBJETIVO.....	1
2. INTRODUÇÃO.....	1
3. FUNDAMENTAÇÃO TEÓRICA.....	1
4. PROGRAMA COMPUTACIONAL.....	2
5. RESULTADOS.....	3

ANEXO

1. OBJETIVO

O objetivo deste trabalho foi implementar algumas rotinas computacionais relacionadas a alguns tópicos apresentados na disciplina de Sistemas Operacionais, principalmente o tópico de thread.

2. INTRODUÇÃO

Enquanto um processo pode ser executado um programa em execução, uma thread pode ser um fluxo único de controle sequencial dentro de um programa. Um programa pode conter apenas uma thread ou mais. Alguns dos problemas clássicos de sistemas operacionais envolvem o funcionamento de threads.

3. FUNDAMENTAÇÃO TEÓRICA

Para ilustrar e implementar os conceitos de threads e semáforos neste trabalho foram considerados e modificados dois problemas clássicos de sistemas operacionais, o de Produtor-consumidor com buffer limitado e o de execução de corrida.

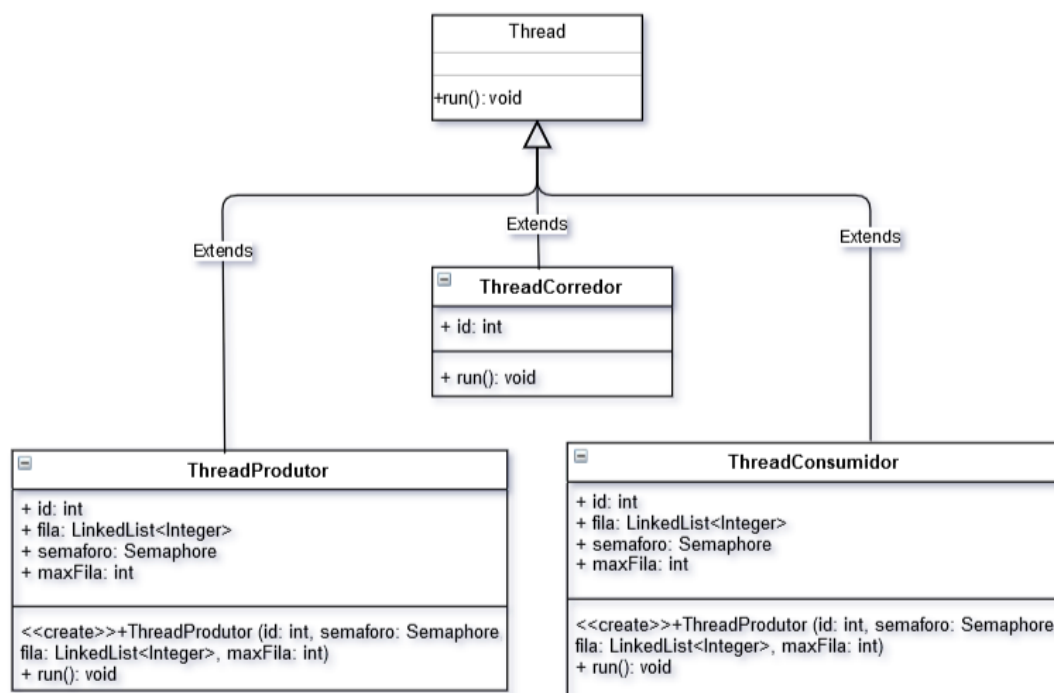
O problema do Produtor-consumidor com buffer limitado considerado foi o cenário em que há uma lista de números sequenciais limitada por um tamanho máximo, onde há um conjunto de threads produtoras que adicionam valores ao final dessa lista, quando ela não está cheia, e um conjunto de threads consumidoras que retiram valores ao final dessa lista, quando ela não está vazia. O objetivo do programa é apenas o de mostrar a pseudoaleatoriedade da ordem de execução das threads, sendo que neste modelo cada uma executa a ação de produzir/consumir apenas uma vez. Foi utilizado um semáforo para delimitar a região crítica.

O problema de execução de corrida foi o cenário em que um determinado número de threads são executados após sua criação, e consistem de uma operação, que no caso é apenas a execução de um loop, e ao finalizar essa operação exibem a mensagem de que finalizaram a corrida. O objetivo do programa é apenas o de mostrar a pseudoaleatoriedade da ordem de execução das threads, sendo que há alteração de execução entre uma thread e outra devido ao tamanho considerável do loop.

4. PROGRAMA COMPUTACIONAL

O programa computacional foi desenvolvido na linguagem Java versão 1.8.

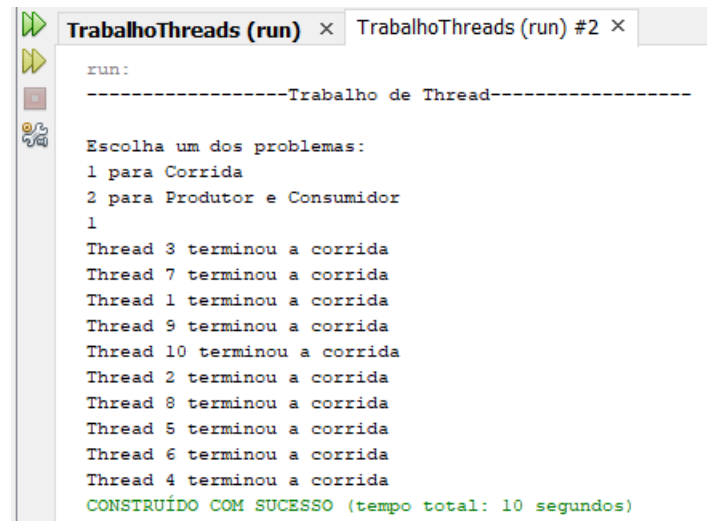
Para implementar threads que executem as funcionalidades desejadas foram criadas classes que herdam da classe Thread nativa da linguagem, e implementam o método run, que contém a sua principal funcionalidade. Foi criada uma classe pra representar cada tipo de thread usada, no caso, Produtor, Consumidor e Corredora. O tipo de “buffer limitado” usado foi uma LinkedList, que apesar de não ser naturalmente limitada, facilitou com seus métodos de remover e adicionar ao final.



Além dessas foi criada a classe estática “TrabalhoThreads” que é classe principal, a qual deve ser executada para que o projeto funcione, é a classe Controller para as outras classes citadas. Ao executar essa classe é pedido ao usuário que digite 1 ou 2 para escolher a execução do problema desejado.

5. RESULTADOS

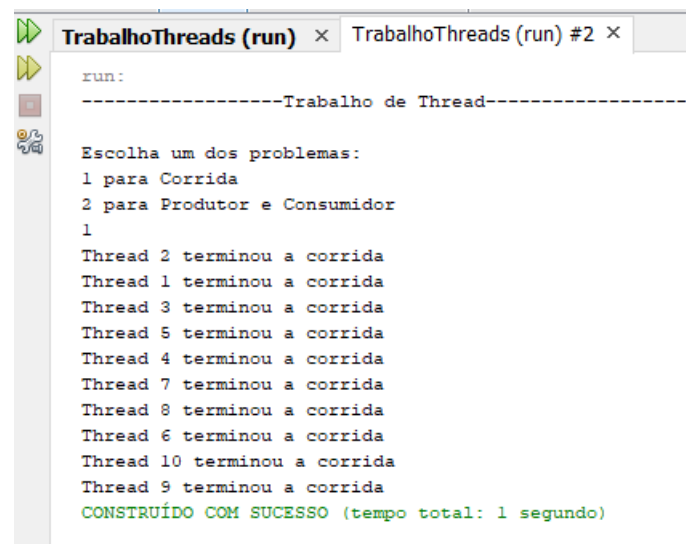
O programa foi executado na IDE NetBeans 8.2, três vezes para a opção 1 (Corrida) e três vezes para a opção 2 (Produtor e Consumidor).



```
run:
-----Trabalho de Thread-----

Escolha um dos problemas:
1 para Corrida
2 para Produtor e Consumidor
1
Thread 3 terminou a corrida
Thread 7 terminou a corrida
Thread 1 terminou a corrida
Thread 9 terminou a corrida
Thread 10 terminou a corrida
Thread 2 terminou a corrida
Thread 8 terminou a corrida
Thread 5 terminou a corrida
Thread 6 terminou a corrida
Thread 4 terminou a corrida
CONSTRUÍDO COM SUCESSO (tempo total: 10 segundos)
```

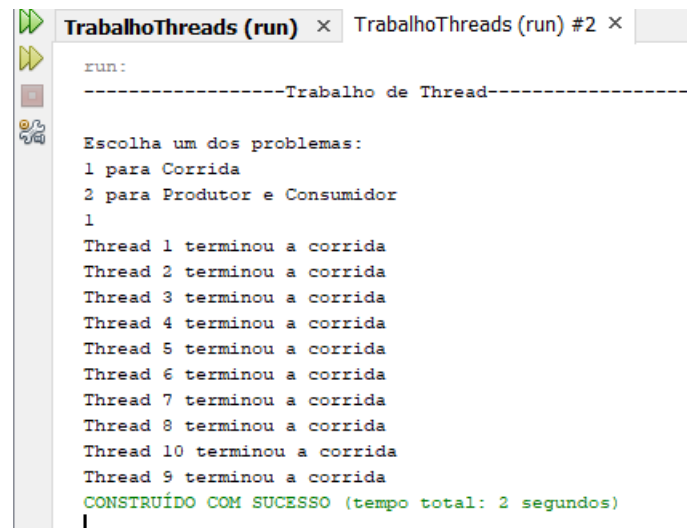
Figura 1 - Execução n° 1 da Corrida



```
run:
-----Trabalho de Thread-----

Escolha um dos problemas:
1 para Corrida
2 para Produtor e Consumidor
1
Thread 2 terminou a corrida
Thread 1 terminou a corrida
Thread 3 terminou a corrida
Thread 5 terminou a corrida
Thread 4 terminou a corrida
Thread 7 terminou a corrida
Thread 8 terminou a corrida
Thread 6 terminou a corrida
Thread 10 terminou a corrida
Thread 9 terminou a corrida
CONSTRUÍDO COM SUCESSO (tempo total: 1 segundo)
```

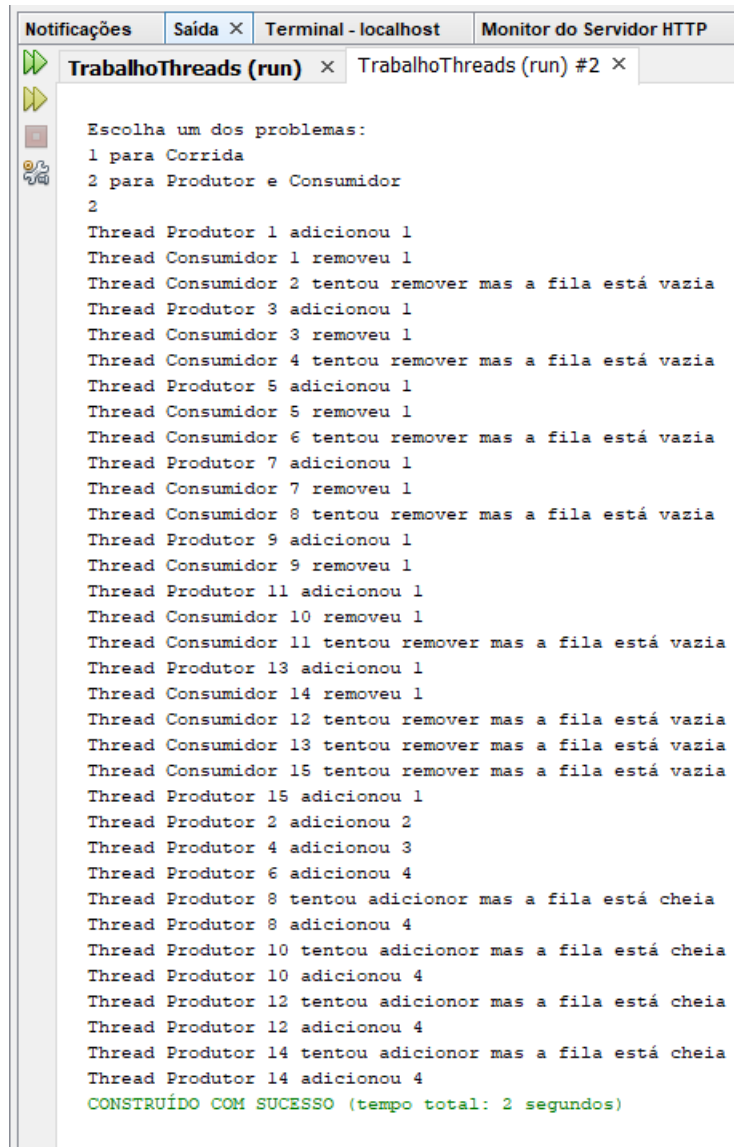
Figura 2 - Execução n° 2 da Corrida



```
run:
-----Trabalho de Thread-----

Escolha um dos problemas:
1 para Corrida
2 para Produtor e Consumidor
1
Thread 1 terminou a corrida
Thread 2 terminou a corrida
Thread 3 terminou a corrida
Thread 4 terminou a corrida
Thread 5 terminou a corrida
Thread 6 terminou a corrida
Thread 7 terminou a corrida
Thread 8 terminou a corrida
Thread 10 terminou a corrida
Thread 9 terminou a corrida
CONSTRUÍDO COM SUCESSO (tempo total: 2 segundos)
```

Figura 3 - Execução n° 3 da Corrida



```

Notificações Saída × Terminal - localhost Monitor do Servidor HTTP
TrabalhoThreads (run) × TrabalhoThreads (run) #2 ×

Escolha um dos problemas:
1 para Corrida
2 para Produtor e Consumidor
2
Thread Produtor 1 adicionou 1
Thread Consumidor 1 removeu 1
Thread Consumidor 2 tentou remover mas a fila está vazia
Thread Produtor 3 adicionou 1
Thread Consumidor 3 removeu 1
Thread Consumidor 4 tentou remover mas a fila está vazia
Thread Produtor 5 adicionou 1
Thread Consumidor 5 removeu 1
Thread Consumidor 6 tentou remover mas a fila está vazia
Thread Produtor 7 adicionou 1
Thread Consumidor 7 removeu 1
Thread Consumidor 8 tentou remover mas a fila está vazia
Thread Produtor 9 adicionou 1
Thread Consumidor 9 removeu 1
Thread Produtor 11 adicionou 1
Thread Consumidor 10 removeu 1
Thread Consumidor 11 tentou remover mas a fila está vazia
Thread Produtor 13 adicionou 1
Thread Consumidor 14 removeu 1
Thread Consumidor 12 tentou remover mas a fila está vazia
Thread Consumidor 13 tentou remover mas a fila está vazia
Thread Consumidor 15 tentou remover mas a fila está vazia
Thread Produtor 15 adicionou 1
Thread Produtor 2 adicionou 2
Thread Produtor 4 adicionou 3
Thread Produtor 6 adicionou 4
Thread Produtor 8 tentou adicionar mas a fila está cheia
Thread Produtor 8 adicionou 4
Thread Produtor 10 tentou adicionar mas a fila está cheia
Thread Produtor 10 adicionou 4
Thread Produtor 12 tentou adicionar mas a fila está cheia
Thread Produtor 12 adicionou 4
Thread Produtor 14 tentou adicionar mas a fila está cheia
Thread Produtor 14 adicionou 4
CONSTRUÍDO COM SUCESSO (tempo total: 2 segundos)

```

Figura 4 - Execução nº 1 do Produtor e Consumidor

```

run:
-----Trabalho de Thread-----

Escolha um dos problemas:
1 para Corrida
2 para Produtor e Consumidor
2

Thread Produtor 1 adicionou 1
Thread Consumidor 1 removeu 1
Thread Produtor 2 adicionou 1
Thread Consumidor 2 removeu 1
Thread Produtor 3 adicionou 1
Thread Consumidor 4 removeu 1
Thread Consumidor 3 tentou remover mas a fila está vazia
Thread Produtor 5 adicionou 1
Thread Consumidor 5 removeu 1
Thread Produtor 6 adicionou 1
Thread Produtor 7 adicionou 2
Thread Consumidor 7 removeu 2
Thread Produtor 8 adicionou 2
Thread Produtor 9 adicionou 3
Thread Consumidor 9 removeu 3
Thread Produtor 10 adicionou 3
Thread Produtor 11 adicionou 4
Thread Consumidor 11 removeu 4
Thread Produtor 12 adicionou 4
Thread Produtor 13 tentou adicionar mas a fila está cheia
Thread Produtor 13 adicionou 4
Thread Consumidor 12 removeu 4
Thread Produtor 14 adicionou 4
Thread Produtor 15 tentou adicionar mas a fila está cheia
Thread Produtor 15 adicionou 4
Thread Consumidor 15 removeu 4
Thread Produtor 4 adicionou 4
Thread Consumidor 6 removeu 4
Thread Consumidor 8 removeu 3
Thread Consumidor 10 removeu 2
Thread Consumidor 14 removeu 1
Thread Consumidor 13 tentou remover mas a fila está vazia
CONSTRUÍDO COM SUCESSO (tempo total: 1 segundo)

```

Figura 5 - Execução nº 2 do Produtor e Consumidor


```

run:
-----Trabalho de Thread-----

Escolha um dos problemas:
1 para Corrida
2 para Produtor e Consumidor
2
Thread Produtor 1 adicionou 1
Thread Consumidor 1 removeu 1
Thread Produtor 2 adicionou 1
Thread Produtor 3 adicionou 2
Thread Produtor 4 adicionou 3
Thread Consumidor 3 removeu 3
Thread Produtor 5 adicionou 3
Thread Consumidor 6 removeu 3
Thread Produtor 7 adicionou 3
Thread Consumidor 5 removeu 3
Thread Consumidor 8 removeu 2
Thread Produtor 9 adicionou 2
Thread Consumidor 10 removeu 2
Thread Consumidor 11 removeu 1
Thread Produtor 12 adicionou 1
Thread Consumidor 12 removeu 1
Thread Consumidor 14 tentou remover mas a fila está vazia
Thread Consumidor 13 tentou remover mas a fila está vazia
Thread Consumidor 2 tentou remover mas a fila está vazia
Thread Consumidor 4 tentou remover mas a fila está vazia
Thread Produtor 6 adicionou 1
Thread Consumidor 7 removeu 1
Thread Produtor 10 adicionou 1
Thread Produtor 11 adicionou 2
Thread Produtor 13 adicionou 3
Thread Consumidor 9 removeu 3
Thread Produtor 14 adicionou 3
Thread Consumidor 15 removeu 3
Thread Produtor 8 adicionou 3
Thread Produtor 15 adicionou 4
CONSTRUÍDO COM SUCESSO (tempo total: 1 segundo)

```

Figura 6 - Execução nº 3 do Produtor e Consumidor

Pelos resultados apresentados, os conceitos e hipóteses de pseudoaleatoriedade da implementação de threads foram comprovados.

ANEXO – Código-fonte

```
package trabalhothreads;

import java.util.LinkedList;

import java.util.Scanner;

import java.util.concurrent.Semaphore;

/**
 *
 * @author Leonardo Simões
 */

public class TrabalhoThreads {

    public static void Corrida(){

        //LinkedList<Integer> fila = new LinkedList<Integer>();

        int numeroDeParesDeProcessos = 10;

        Thread[] processos = new Thread[numeroDeParesDeProcessos];

        for (int i = 0; i < numeroDeParesDeProcessos; i++) {

            processos[i] = new ThreadCorredora(i + 1);

            processos[i].start();

        }

    }

    public static void ProdutorConsumidor() {

        LinkedList<Integer> fila = new LinkedList<Integer>();

        int maxFila = 4;

        Semaphore semaforo = new Semaphore(1);

        int numeroDeParesDeProcessos = 15;

        Thread[] processos = new Thread[numeroDeParesDeProcessos];
```

```

for (int i = 0; i < numeroDeParesDeProcessos; ++i) {

    processoss[i] = new ThreadProdutor(i + 1, semaforo, fila, maxFila);

    processoss[i].start();

    processoss[i] = new ThreadConsumidor(i + 1, semaforo, fila, maxFila);

    processoss[i].start();

}

}

/**

 * @param args the command line arguments

 */

public static void main(String[] args) {

    // TODO code application logic here

    int opcao = 0;

    Scanner entrada = new Scanner(System.in);

    do{

        System.out.println("-----Trabalho de Thread-----");

        System.out.println(" \nEscolha um dos problemas:");

        System.out.println("1 para Corrida \n2 para Produtor e Consumidor");

        //Entrada do valor da opcao

        try{

            opcao = entrada.nextInt();

        }catch(Exception e){

            System.out.println("");

        }

        //Execucao da opercao selecionada

        if(opcao==1){

            Corrida();

```

```

        }else if(opcao==2){
            ProdutorConsumidor();
        }
    }while(opcao!=1 && opcao!=2);
}
}

```

```

package trabalhothreads;

import java.util.LinkedList;

/**
 *
 * @author Leonardo Simões
 */
public class ThreadCorredora extends Thread{

    private int id;

    public ThreadCorredora(int id) {

        this.id = id;

    }

    public void run() {

        for(int i=0; i<5000; ++i){

            //Corrida

        }

        System.out.println("Thread " + this.id + " terminou a corrida");

    }

}

```

```

package trabalhothreads;

import java.util.LinkedList;

import java.util.concurrent.Semaphore;

/**
 *
 * @author Leonardo Simões
 */

public class ThreadProdutor extends Thread {

    private int id;

    private LinkedList<Integer> fila;

    private Semaphore semaforo;

    private final int maxFila;

    public ThreadProdutor(int id, Semaphore semaforo, LinkedList<Integer> fila, int
maxFila) {

        this.id = id;

        this.fila = fila;

        this.maxFila = maxFila;

        this.semaforo = semaforo;

    }

    public void run() {

        try {

            semaforo.acquire();

            if (this.fila.size() >= this.maxFila) {

                System.out.println("Thread Produtor " + this.id + " tentou adicionar mas a fila
está cheia");

            } else if (this.fila.isEmpty()) {

                this.fila.add(new Integer(1));

            } else {

```

```

        this.fila.add(this.fila.getLast() + 1);
    }

    System.out.println("Thread Produtor " + this.id + " adicionou " +
this.fila.getLast());

    } catch (InterruptedException e) {

    } finally {

        semaforo.release();

    }

}

}

```

```

package trabalhothreads;

import java.util.LinkedList;

import java.util.concurrent.Semaphore;

/**
 *
 * @author Leonardo Simões
 */

public class ThreadConsumidor extends Thread {

    private int id;

    private LinkedList<Integer> fila;

    private final int maxFila;

    private Semaphore semaforo;

    public ThreadConsumidor(int id, Semaphore semaforo, LinkedList<Integer> fila, int
maxFila) {

        this.id = id;

        this.semaforo = semaforo;

        this.fila = fila;

```

```
        this.maxFila = maxFila;
    }

    public void run() {
        try {
            semaforo.acquire();

            if (!this.fila.isEmpty()) {

                System.out.println("Thread Consumidor " + this.id + " removeu " +
this.fila.removeLast());

            }else{

                System.out.println("Thread Consumidor " + this.id + " tentou remover mas a
fila está vazia");

            }

        } catch (InterruptedException e) {

        }finally{

            semaforo.release();

        }

    }

}
```