Aula prática 3

Monitoria InfraCom 2020.3

Josenildo Vicente (jva)

Objetivo

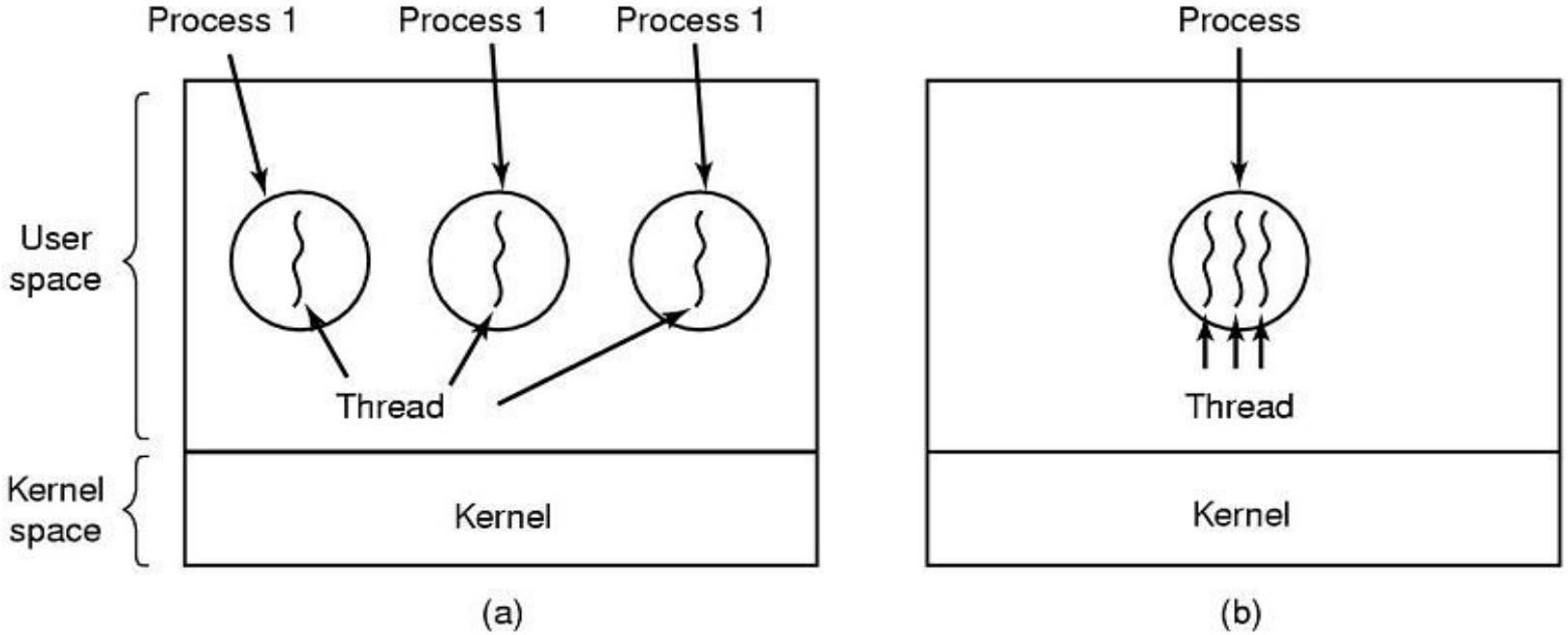
- Programação Concorrente
- P2P

Programação Concorrente

É um paradigma de programação usado na construção de programas que fazem uso da execução concorrente de várias tarefas computacionais interativas, que podem ser implementadas como programas separados ou como um conjunto de **threads** criadas por um único programa.

Designa a programação paralela e a programação distribuída.

Threads


Process 1
Process 1
Process 1
User space
Thread
Kernel space
Kernel
(a)
Process
Thread
Kernel
(b)

Estados de uma thread

Criação: Neste estado, o processo pai está criando a thread que é levada a fila de prontos;

Execução: Neste estado a thread está usando a CPU;

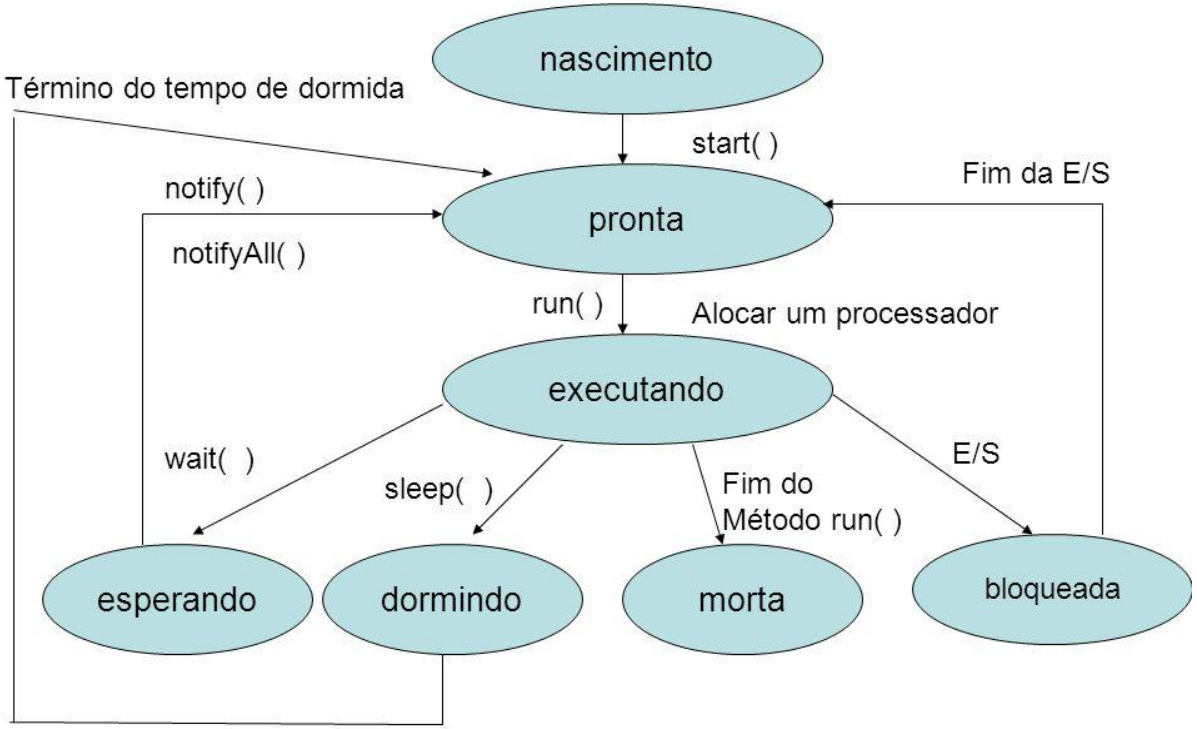
Pronto: Neste estado a thread avisa a CPU que pode entrar no estado de execução e entra na fila de prontos;

Bloqueado: Neste estado, por algum motivo, a CPU bloqueia a thread, geralmente enquanto aguarda algum dispositivo de I/O;

Término: Neste estado são desativados o contexto de hardware e a pilha é desalocada.

Esperando e Finalizado.

Estados de uma thread



Métodos em Java

start(): inicia a execução da thread (método run)

suspend(): suspende a execução da thread que está executando

sleep(): faz a thread que está executando dormir por um tempo determinado

yield(): faz a thread que está executando dormir por um tempo indeterminado

resume(): resume a execução de uma thread suspensa

stop(): termina a execução de uma thread; a thread não pode ser mais executada.

Métodos em Java

join(): método que espera o término da(s) THREAD(S) que estão sendo executadas.

interrupt(): método que interrompe a execução de uma THREAD.

interrupted(): método que testa se a THREAD atual está ou não interrompida. (estático)

isInterrupted(): método que testa se o objeto em que foi chamado está ou não interrompido.

https://docs.oracle.com/javase/9/docs/api/java/lang/Thread.html

Exemplo
extends Thread

```
class threadSimples extends Thread{
    public threadSimples(String nome) {
        super(nome);
    }
    public void run() {
        System.out.println("Thread " + getName() + " comecou a contar!!");
        for(int i =0; i<=10; i++ ) {
            System.out.println( i + "   " + getName());
        }
        System.out.println("Thread " + getName() + " Acabou de contar!!");
    }
}

public class TesteConcorrenciaDeThreads {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        new threadSimples("Primeiro").start();
        new threadSimples("Segundo").start();
        new threadSimples("Terceiro").start();
        new threadSimples("Quarto").start();
    }

}
```

Exemplo
runnable

```java
public class TesteConcorrenciaRunnable {

    public static void main(String[] args) {

        new Thread(primeiro).start();
        new Thread(segundo).start();

    }

    private static Runnable primeiro = new Runnable() {
        public void run() {
            System.out.println("Runnable primeiro comecou a contar!!");
            for(int i =0; i<=10; i++) {
                System.out.println( i + "  primeiro");
            }
            System.out.println("Runnable primeiro acabou de contar!!");
        }
    };

    private static Runnable segundo = new Runnable() {
        public void run() {
            System.out.println("Runnable segundo comecou a contar!!");
            for(int i =0; i<=10; i++) {
                System.out.println( i + "  segundo");
            }
            System.out.println("Runnable segundo acabou de contar!!");
        }
    };

}
```

Exemplo

runnable contador

```
public class TesteConcorrenciaRunnableContador {

    static int i = 0;
    public static void main(String[] args) {
        new Thread(primeiro).start();
        new Thread(segundo).start();
    }

    private static boolean countMe(String name){
        if(i > 10 ) {
            return false;
        }
        System.out.println("Contador atualmente esta em: " + i++ + ", atualizado por: " + name);
        return true;
    }
```

Exemplo

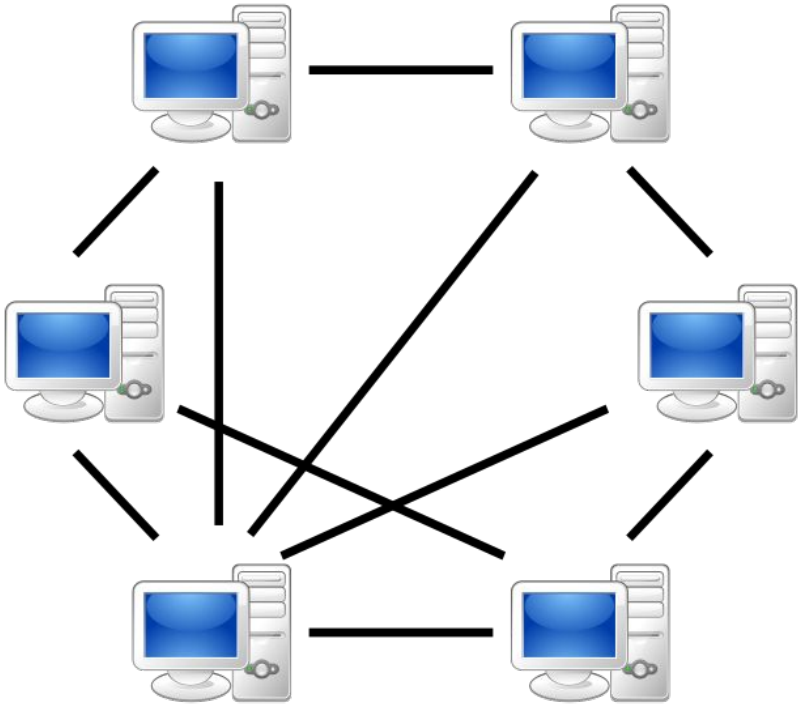
runnable contador

```
    private static Runnable primeiro = new Runnable() {
        public void run() {
            boolean cont=true;
            try{
                while(cont){
                    cont=countMe("primeiro");
                    if(!cont) {
                        break;
                    }
                }
            } catch (Exception e){}

        }
    };

    private static Runnable segundo = new Runnable() {
        public void run() {
            boolean cont=true;
            try{
                while(cont){
                    cont = countMe("segundo");
                    if(!cont) {
                        break;
                    }
                }
            } catch (Exception e){}
       }
    };

}
```

P2P

Peer to Peer

É uma arquitetura de redes de computadores onde cada um dos pontos ou nós da rede funciona tanto como cliente quanto como servidor, permitindo compartilhamentos de serviços e dados sem a necessidade de um servidor central.



P2P Híbrido

- Por que ter um servidor?
  - Associação entre usuário e seu endereço IP;
  - Guardar mensagens enviadas para usuários offline;
  - E mais importante: resolver o problema de NAT (ainda não visto nas aulas teóricas);

Lista de exercício 3

Fazer lista 3 listada no classroom

Acompanhamento durante a aula