MÁQUINA VIRTUAL JAVA

Processos, Threads, Semáforos, Mutexes e compartilhamento de memória

Daniele Mendonça de Carvalho
Marcos Vinicio Araujo Delgado Junior
Weverson Paulo SIlva FIlho

THREADS

A JVM permite uma aplicação possuir diversas threads em execução

Podemos utilizar de diversas maneiras:

- Maneira mais simples:

```
new Thread(){
    @Override
    public void run() {
        // Função a rodar na Thread
    }
}.start();
```

Obs.: Precisamos sempre implementar o Método run(), se quisermos adicionar uma função para rodar na thread em paralelo.

Podemos dar um @Override tanto na própria classe Thread, como podemos utilizar a interface ***Runnable***.

```
Runnable runnable = new Runnable(){
    @Override
    public void run() {
        // Função a rodar na Thread
    }
};
```

E em seguida:

```
new Thread(runnable).start();
```

Extends Thread vs Implements Runnable

Quando você utiliza o extend Thread:

- Você não pode extender de outra classe.
- Não possui reusabilidade de código.
- Manutenção do código não é muito boa

Quando você utiliza o implements Runnable:

- Você pode extender de outra classe
- Possui reusabilidade de código
- Manutenção de código é bem mais fácil.

1° exemplo

```
public class SimpleThread {
    public static void main(String[] args) {
        new Thread(runnable).start();
        while (true){
            System.out.println("Sou o processo principal");
        }
    }

    private static Runnable runnable = new Runnable() {
        @Override
        public void run() {
            while (true){
                System.out.println("Sou o processo em paralelo");
            }
        }
    };
}
```

2° exemplo

SimpleThread.java:

```
public class SimpleThread {
    public static void main(String[] args) {
        new ThreadTest().start();
        while (true){
            System.out.println("Sou o processo principal");
        }
    }
}
```

ThreadTest.java

```
public class ThreadTest extends Thread {
    @Override
    public void run() {
        while (true){
            System.out.println("Sou o processo em
paralelo");
        }
    }
}
```

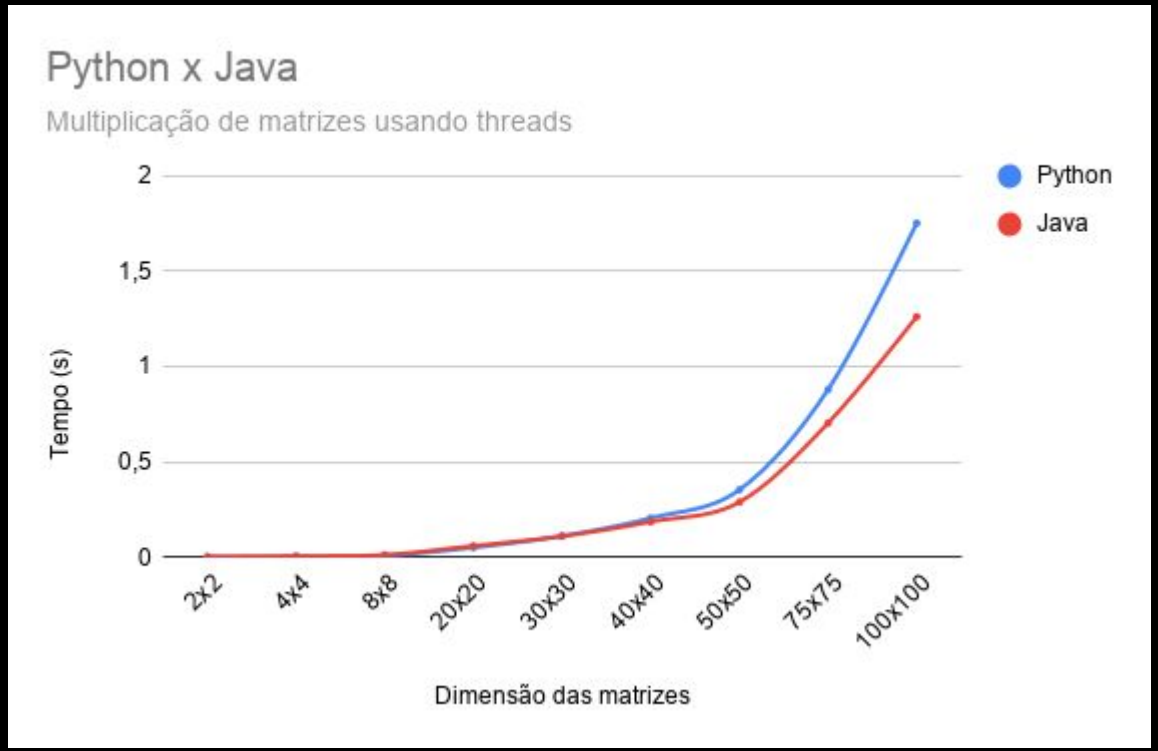
3° exemplo

```
public class SimpleThread2 {
    public static void main(String[] args) throws InterruptedException {
   Thread thread = new Thread() {
     @Override
     public void run() {
       while (true) {
          System.out.println("Processo em paralelo");
          try {
            sleep(1000);
          }
          catch (InterruptedException e) {
             e.printStackTrace();
          }
       }
     }
  };
      thread.start();
            while(true){
               System.out.println("Processo
    principal");
               Thread.sleep(1000);
            }
        }
    }
```

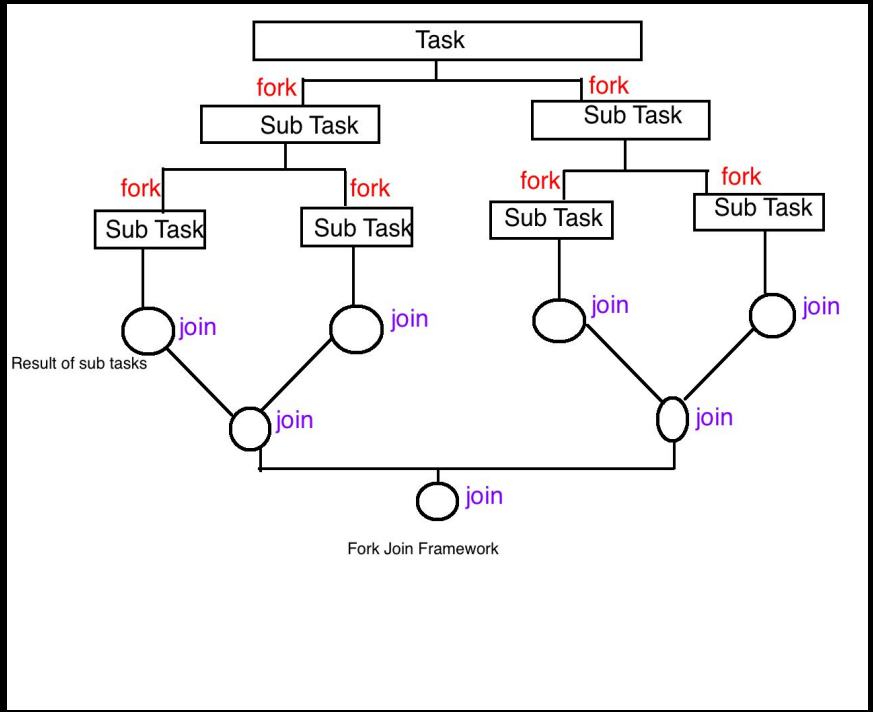
Exemplo em Python

```
import threading
import time
import os

def thread_func():
    t = threading.currentThread()
    while True :
        print ("Sou o processo em paralelo")
        time.sleep(1)


x = threading.Thread(target=thread_func, args=())
x.start()
while True :
    print ("Sou o processo principal ")
    time.sleep(1)

#x.join()
```

Comparação entre java e python



ForkJoin FrameWork




```java
import java.util.concurrent.*;

public class Fibonacci extends RecursiveTask<Integer> {

   public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        ForkJoinPool pool = new ForkJoinPool();
       ForkJoinTask<Integer> task = new Fibonacci(n);
        int result = pool.invoke(task);
   }

   private final int n;

   public Fibonacci(int n) { this.n = n; }

   public Integer compute() {
        if (n <= 1) return n;
        Fibonacci f1 = new Fibonacci(n - 1);
        f1.fork();
        Fibonacci f2 = new Fibonacci(n - 2);
        return f2.invoke() + f1.join();
   }
}
```

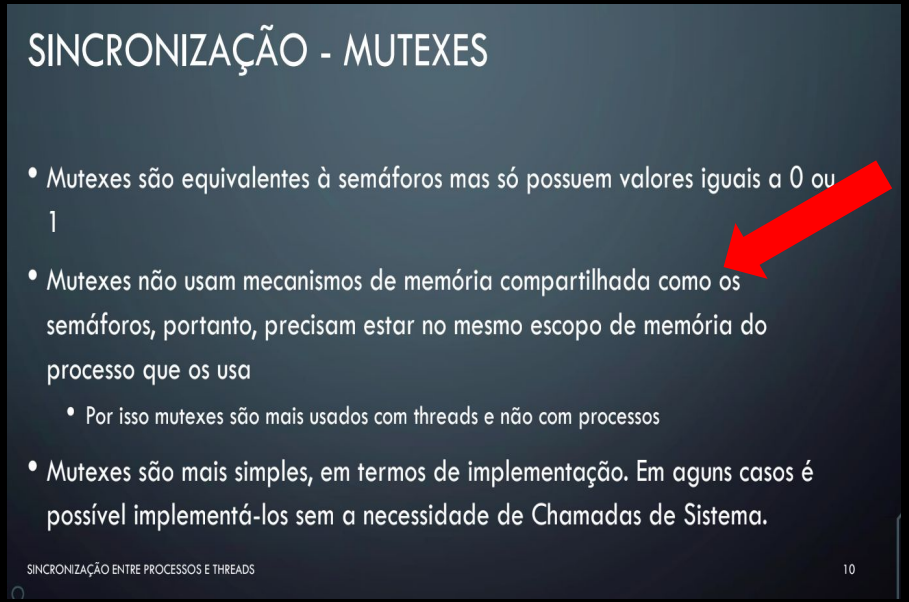
PROCESSOS

- ProcessBuilder
- RunTime
- Process

```
public class testeBuilder {

    public static void main(String[] args) {
        try {
            // create a new process
            System.out.println("Creating Process");

            ProcessBuilder builder = new ProcessBuilder("gedit");
            Process pro = builder.start();

            // wait 10 seconds
            System.out.println("Waiting");
            Thread.sleep(10000);

            // kill the process
            pro.destroy();
            System.out.println("Process destroyed");
        }
            catch (Exception ex)  {
            ex.printStackTrace();
        }
    }
}
```

SEMÁFOROS e MUTEXES

Vamos relembrar:



Em java possuímos a classe Semaphore, porém apesar do nome, essa classe acaba funcionando como um Mutex, pois ela não utiliza recursos de memória compartilhada, além disso essa classe é utilizada com threads e não processos.

Porém a quantidade de threads que podem exercer ações durante um *locking de uma thread específica* pode ir além de 1 ou 0.

Exemplo:

```java
import java.util.concurrent.Semaphore;

public class SemaphoreExample extends Thread{ // poderiamos criar uma classe estática dentro da classe SemaphoreExample
    private static Semaphore semaphore = new Semaphore(1);
    private String threadId;


    public SemaphoreExample(String threadId) {
        this.threadId = threadId;
    }

    @Override
    public void run() {
        try {
            System.out.println(threadId + " Locking...");
            semaphore.acquire();

            try {
                for(int i=0;i<5;i++){ // vai ficar coisando por 5 segundos
                    System.out.println(threadId + " está realizando uma ação, Avalible permits: " + semaphore.availablePermits());
                    sleep(1000);
                }
            }
            finally { // caso as ações dentro da thread ocorram corretamente
                System.out.println("Realeasing " + threadId);
                semaphore.release();
            }

        }
        catch (InterruptedException e) {
            e.printStackTrace();
        }

    }
```

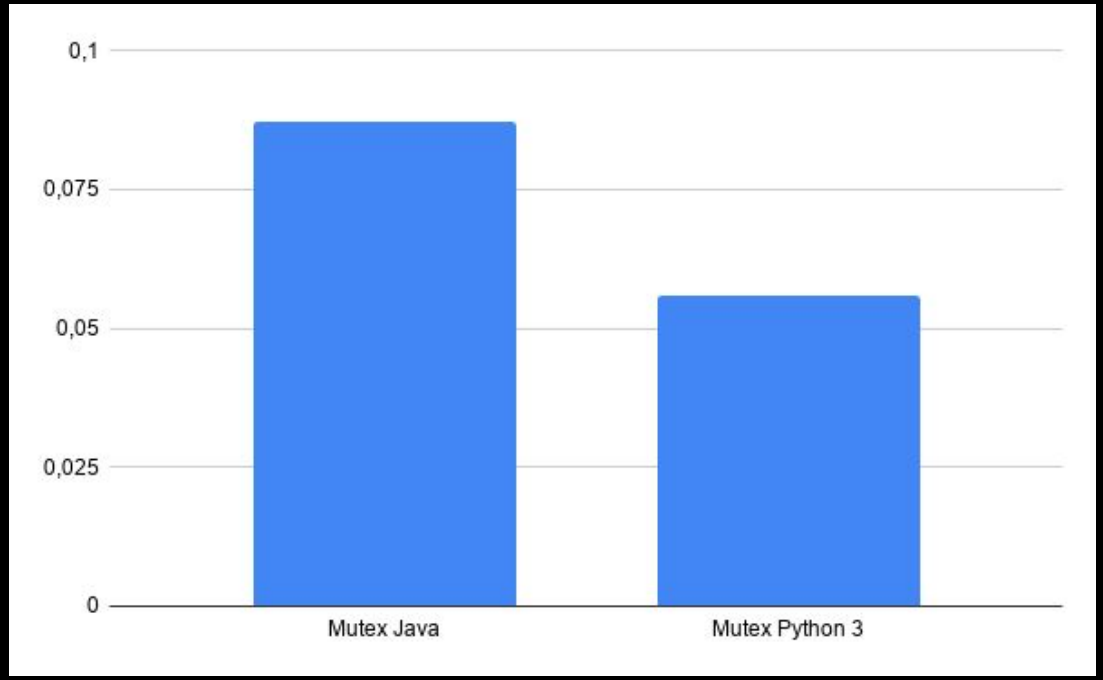
```
    Run | Debug
    public static void main(String[] args) {

        SemaphoreExample t1 = new SemaphoreExample("t1");
        t1.start();

        SemaphoreExample t2 = new SemaphoreExample("t2");
        t2.start();

        SemaphoreExample t3 = new SemaphoreExample("t3");
        t3.start();

        SemaphoreExample t4 = new SemaphoreExample("t4");
        t4.start();

    }


}
```

Exemplo em Python:

```python
import threading
import time
import os

lock = threading.Lock()

class MutexExample:
    def __init__(self, id):
        self.threadid = id
        self.t = threading.Thread(target=self.thread_func, args=())

    def thread_func(self):
        global lock
        print("Locking ", self.threadid)
        lock.acquire()
        for i in range(0,5):
            print(self.threadid, " Está realizando operações")
            time.sleep(1)
        print("Realsing ", self.threadid)
        lock.release()
```

```python
if __name__== "__main__":

    t1 = MutexExample("t1")
    t1.t.start()

    t2 = MutexExample("t2")
    t2.t.start()

    t3 = MutexExample("t3")
    t3.t.start()

    t4 = MutexExample("t4")
    t4.t.start()
```

Comparação entre java e python - Mutex



COMPARTILHAMENTO DE MEMÓRIA

Uma das maneiras de fazer uma comunicação entre processos em java é através de compartilhamento de memória, e uma das maneiras de utilizar o compartilhamento de memória é através de **Memory Mapped Files.**

Memory mapped I/O utiliza o sistema de arquivos para estabelecer um mapeamento de memória virtual.

Com esse recurso nós podemos entender que o arquivo inteiro que você está escrevendo está na memória, e que nós podemos acessá-lo como um array gigante.

Em java nós possuímos a classe **MappedByteBuffer**, que servirá de buffer para podermos escrever e ler informações do Memory Mapped File, além de algumas outras classes para conseguirmos fazer todo o processo.

Exemplo de Escrita:

```java
import java.io.File;
import java.io.RandomAccessFile;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;

public class WriteExample {
    private static String bigTextFile = "test.txt";

    Run | Debug
    public static void main(String[] args) throws Exception {
        File file = new File(bigTextFile);

        //deletamos o arquivo, pois vamos acessar-lo de outra maneira
        file.delete();

        try (RandomAccessFile randomAccessFile = new RandomAccessFile(file, "rw")) {
            FileChannel fileChannel = randomAccessFile.getChannel();

            // Get direct byte buffer access using channel.map() operation
            MappedByteBuffer buffer = fileChannel.map(FileChannel.MapMode.READ_WRITE, 0, 4096 * 8 * 8);

            buffer.put("Escrevendo em um Buffer de saida".getBytes());

            fileChannel.close();
        }
    }
}
```

Exemplo de Leitura:

```java
import java.io.File;
import java.io.RandomAccessFile;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;

public class ReadExample {
    private static String bigExcelFile = "test.txt";

    Run | Debug
    public static void main(String[] args) throws Exception {
        try (RandomAccessFile file = new RandomAccessFile(new File(bigExcelFile), "r")) {
            FileChannel fileChannel = file.getChannel();

            // Get direct byte buffer access using channel.map() operation
            MappedByteBuffer buffer = fileChannel.map(FileChannel.MapMode.READ_ONLY, 0, fileChannel.size());

            System.out.println("Is Lodaded? -> " + buffer.isLoaded());
            System.out.println("Buffer capacity: " + buffer.capacity());

            for (int i = 0; i < buffer.limit(); i++) {
                System.out.print((char) buffer.get());
            }
            System.out.print("\n");

            fileChannel.close();
        }
    }
}
```

- Note que também é necessário utilizar a classe **RandomAccessFile**, permitindo que possamos escrever e ler em um arquivos de acesso randômico. É utilizada essa classe pois um arquivo de acesso randômico se comporta como um array gigante armazenado em um arquivo. Comentamos esse conceito anteriormente.

- Além disso necessitamos da Classe **FileChannel** para mapear e manipular os bytes em arquivos

Exemplo de escrita e leitura em Python:

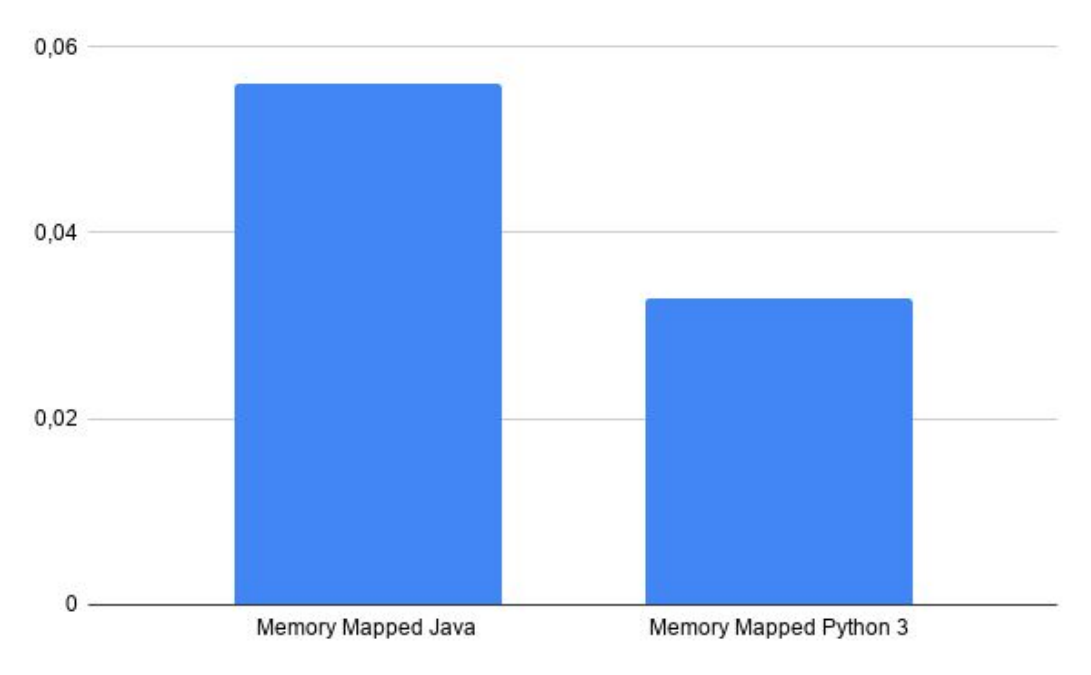
Escrita:

```
with open("hello.txt", "wb") as f:
   f.write(b"Escrevendo em um buffer de saida\n")
```

Leitura:

```
import mmap

with open("hello.txt", "r+b") as f:
   # memory-map the file, size 0 means whole file
   mm = mmap.mmap(f.fileno(), 0)
   # read content via standard file methods
   # ... and read again using standard file methods
   line = mm.readline()
   print(line)  # prints b"Hello  world!\n"
   # close the map
   mm.close()
```

Comparação entre java e python
Memory Mapped File



FIM