UNIVERSIDADE FEDERAL DE LAVRAS
Departamento de Computação Aplicada

REO 8 - TEXTO COMPARATIVO ENTRE CONCORRÊNCIA EM JAVA THREADS E POSIX THREADS

Código e Nome:GCC177 – Programação Paralela e Concorrente
Nome do Aluno: Mateus Carvalho Gonçalves - 201810245

O desenvolvimento da linguagem Java (inicialmente chamada de “Oak”) se iniciou como parte do **Projeto Green** em 1991, encabeçado por James Gosling. O principal objetivo do projeto era o “Write Once, Run Anywhere” (“Escreva uma vez, execute em qualquer lugar” - ou seja, portabilidade) para comunicação entre eletrônicos. Para isso, seria necessário uma máquina virtual e uma linguagem para ela. Essa linguagem deveria ter uma notação similar ao C, porém mais uniforme e simples, e suporte à orientação a objetos, que hoje é seu principal paradigma [1, 2].

Sua primeira implementação pública foi apenas em 1995, em que a Sun viu uma oportunidade na web para melhorar a interatividade das páginas e resultou em um absoluto sucesso, gerando uma aceitação aos browsers populares como o **Netscape Navigator** e padrões tridimensionais como o **VRML** (Virtual Reality Modeling Language, ou Linguagem de Modelagem para Realidade Virtual em português) [1].

O processo de compilação e execução de códigos Java é diferente dos métodos usuais, uma vez que utiliza uma máquina virtual para possibilitar a independência de máquina. São necessárias 3 ferramentas para compilar e executar um código Java: (1) o **JDK** - Java Development Kit - é o Kit de Desenvolvimento Java responsável por compilar código-fonte (.java) em bytecode (.class); (2) a **JVM** - (Java Virtual Machine) - é a Máquina Virtual do Java responsável por executar o bytecode (.class); e (3) **JRE** (Java Runtime Environment), ou Ambiente de Execução do Java, fornece as bibliotecas padrões do Java para o JDK compilar o seu código e para a JVM executar o seu programa [3].

Java oferece suporte a concorrência por meio do pacote *java.util.concurrent* e outras classes e interfaces do pacote padrão (*java.lang*), como a classe **Thread** por exemplo. Também é possível criar processos no Java por meio da classe ProcessBuilder.

Uma diferença importante já para criar uma thread a partir dessas duas ferramentas é que com a função `pthread_create()` a thread executa de imediato, enquanto com Java é necessário expressar o comando `start()`, entre outras funções, isso possibilita que mais parâmetros sejam setados antes da execução [4]. Porém, as duas implementações têm características semelhantes, como passagem por parâmetro das funções que a nova thread deve executar, suporte à sincronização, etc.

Para criação básica de uma thread em Java existem dois principais métodos: criação de classe que herda da classe Thread, ou que implementa a interface Runnable [5]. A Tabela 1 apresenta um exemplo básico utilizando o primeiro método. Já a Figura 1 mostra um programa simples utilizando pthreads [6].

Tabela 1. Exemplo de criação de thread em Java herdando da classe thread

```
1   class PrimeThread extends Thread {
2       long minPrime;
3       PrimeThread(long minPrime) {
4            this.minPrime = minPrime;
5       }
6
7       public void run() {
8           // compute primes larger than minPrime
9           . . .
10      }
11  }
12
13  ...
14  PrimeRun p = new PrimeRun(143);
15  new Thread(p).start();
16  ...
```

Figura 1. Exemplo de criação de threads com pthread_create() e terminação com pthread_exit()

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS     5

void *PrintHello(void *threadid)
{
   long tid;
   tid = (long)threadid;
   printf("Hello World! It's me, thread #%ld!\n", tid);
   pthread_exit(NULL);
}

int main (int argc, char *argv[])
{
   pthread_t threads[NUM_THREADS];
   int rc;
   long t;
   for(t=0; t<NUM_THREADS; t++){
      printf("In main: creating thread %ld\n", t);
      rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
      if (rc){
         printf("ERROR; return code from pthread_create() is %d\n", rc);
         exit(-1);
      }
   }

   /* Last thing that main() should do */
   pthread_exit(NULL);
}
```

No Java também é possível obter o identificador de uma thread a partir da sequência de chamadas da classe Thread: `Thread.currentThread().getId()` [7]. A Tabela 2 contém um exemplo de código que obtém e printa os nomes e identificadores de 2 threads criadas.

Tanto nas pthreads quanto no Java é possível fazer comunicação entre processos. A implementação de Pipes segue linhas de raciocínio similares, como mostrado nas Figuras 2, 3, 4 e 5, do problema Produtor-Consumidor. Outras implementações mais comuns são por Sockets, RMI e CORBA, e cada uma possui uma maneira distinta de implementar a comunicação [8].

Tabela 2. Código para obter identificador da thread usa sincronização por join

```
1   // Java program to get the id of a thread
2
3   import java.util.*;
4   public class ThreadDemo1 extends Thread {
5       public void run() {
6           // gets the name of current thread
7           System.out.println("Current Thread Name: "
8             + Thread.currentThread().getName());
9
10          // gets the ID of the current thread
11          System.out.println("Current Thread ID: "
12            + Thread.currentThread().getId());
13      }
14
15      public static void main(String[] args)
16        throws InterruptedException {
17          Scanner s = new Scanner(System.in);
18
19          // creating first thread
20          ThreadDemo1 t1 = new ThreadDemo1();
21
22          // creating second thread
23          ThreadDemo1 t2 = new ThreadDemo1();
24
25          // Starting the thread
26          t1.start();
27
28          // t2 does not start execution until t1 completes execution
29          t1.join();
30          t2.start();
31      }
32  }
```

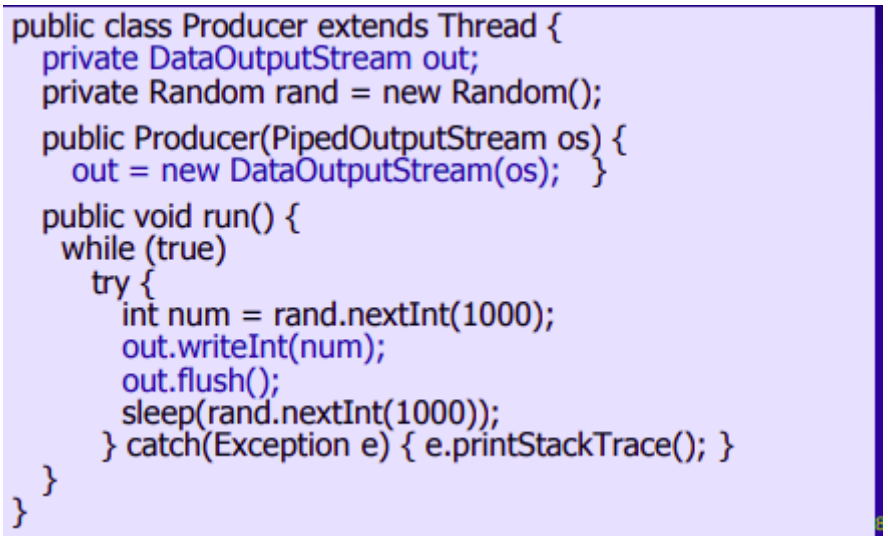
Figura 2. Classe principal para comunicação por Pipes em Java

```
public class PipeTest {
  public static void main(String args[]) {
    try {
      PipedOutputStream out = new PipedOutputStream();
      PipedInputStream in = new PipedInputStream(out);

      Producer prod = new Producer(out);
      Consumer cons = new Consumer(in);

      prod.start();
      cons.start();
    } catch (IOException e) { e.printStackTrace(); }
  }
}
```

Figura 3. Classe consumidora para comunicação por Pipes em Java

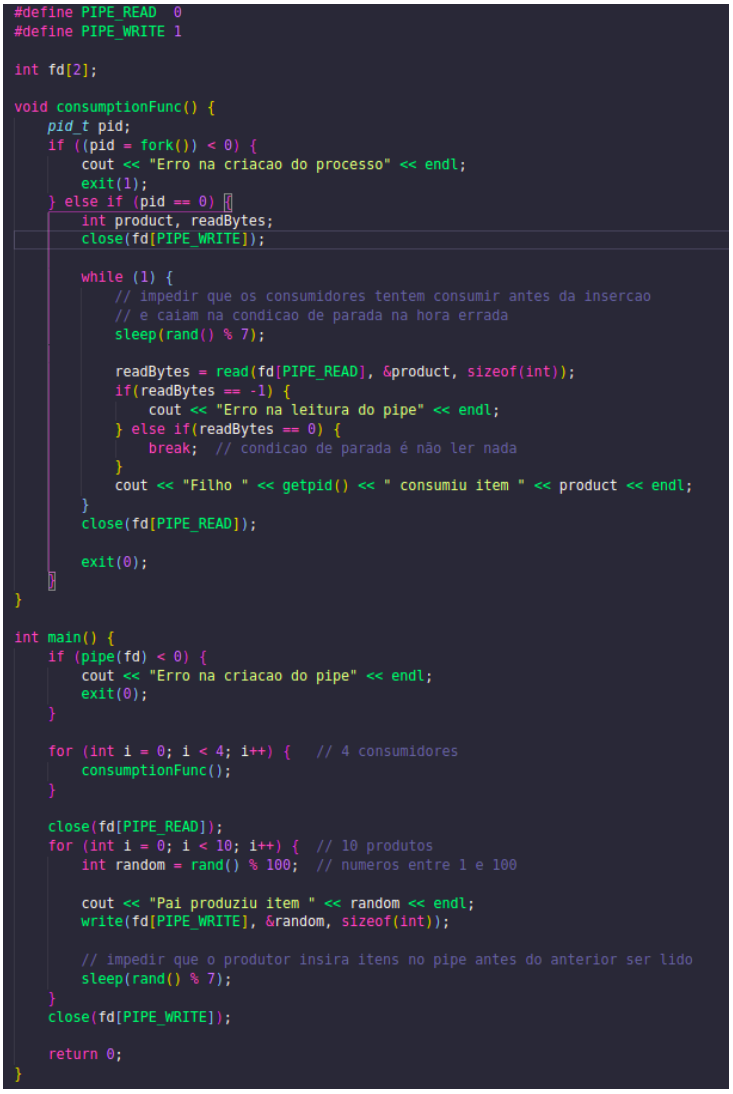
```
public class Consumer extends Thread {
  private DataInputStream in;
  public Consumer(PipedInputStream is) {
    in = new DataInputStream(is); }
  public void run() {
    while(true)
      try {
        int num = in.readInt();
        System.out.println("Número recebido: " + num);
      } catch(IOException e) {  e.printStackTrace(); }
  }
}
```

Figura 4. Classe produtora para comunicação por Pipes em Java



```
public class Producer extends Thread {
  private DataOutputStream out;
  private Random rand = new Random();
  public Producer(PipedOutputStream os) {
    out = new DataOutputStream(os);   }
  public void run() {
   while (true)
     try {
       int num = rand.nextInt(1000);
       out.writeInt(num);
       out.flush();
       sleep(rand.nextInt(1000));
     } catch(Exception e) { e.printStackTrace(); }
  }
}
```

Figura 5. Comunicação por pipes em pthread (C++)



```
#define PIPE_READ  0
#define PIPE_WRITE 1

int fd[2];

void consumptionFunc() {
    pid_t pid;
    if ((pid = fork()) < 0) {
        cout << "Erro na criacao do processo" << endl;
        exit(1);
    } else if (pid == 0) {
        int product, readBytes;
        close(fd[PIPE_WRITE]);

        while (1) {
            // impedir que os consumidores tentem consumir antes da insercao
            // e caiam na condicao de parada na hora errada
            sleep(rand() % 7);

            readBytes = read(fd[PIPE_READ], &product, sizeof(int));
            if(readBytes == -1) {
                cout << "Erro na leitura do pipe" << endl;
            } else if(readBytes == 0) {
                break;  // condicao de parada é não ler nada
            }
            cout << "Filho " << getpid() << " consumiu item " << product << endl;
        }
        close(fd[PIPE_READ]);

        exit(0);
    }
}

int main() {
    if (pipe(fd) < 0) {
        cout << "Erro na criacao do pipe" << endl;
        exit(0);
    }

    for (int i = 0; i < 4; i++) {   // 4 consumidores
        consumptionFunc();
    }

    close(fd[PIPE_READ]);
    for (int i = 0; i < 10; i++) {  // 10 produtos
        int random = rand() % 100;  // numeros entre 1 e 100

        cout << "Pai produziu item " << random << endl;
        write(fd[PIPE_WRITE], &random, sizeof(int));

        // impedir que o produtor insira itens no pipe antes do anterior ser lido
        sleep(rand() % 7);
    }
    close(fd[PIPE_WRITE]);

    return 0;
}
```

Por fim, enquanto as bibliotecas pthreads oferecem técnicas de sincronização diversas formas, entre elas o semáforo mutex e com variável de condição. De maneira similar, em Java temos as implementações *synchronized*, com suporte das funções *wait()*, *notify()* e *notifyAll()* [5, 9, 10]. E também as implementações *lock*, além de técnicas de sincronização implícitas, como o *join()* (Tabela 2).

Tabela 3. Implementação básica de um semáforo synchonized (similar ao semáforo com variável de condição na biblioteca pthread)

```
 1  public class Semaphore {
 2      private boolean signal = false;
 3
 4      public synchronized void take() {
 5          this.signal = true;
 6          this.notify();
 7      }
 8
 9      public synchronized void release() throws InterruptedException{
10          while(!this.signal) wait();
11          this.signal = false;
12      }
13  }
```

REFERÊNCIAS BIBLIOGRÁFICAS

[1] DEVMEDIA. Java: história e principais conceitos. Disponível em https://www.devmedia.com.br/java-historia-e-principais-conceitos/25178. Acesso em 20/03/2021.

[2] Free Java Guide & Tutorials. History of Java programming language. Disponível em https://www.freejavaguide.com/history.html#:~:text=Java%20was%20started%20as%20a,was%20Java%201.0%20in%201995.. Acesso em 20/03/2021.

[3] Java. TechInfo. Disponível em https://www.java.com/pt-BR/download/help/techinfo.html. Acesso em 21/03/2021.

[4] stackoverflow. Java Threads vs Pthreads. Disponível em https://stackoverflow.com/questions/5269535/java-threads-vs-pthreads. Acesso em 21/03/2021.

[5] Java Documentation: Lesson Concorrency. Disponível em https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html. Acesso em 22/03/2021.

[6] Lawrence Livermore National Laboratory. Creating and Terminating Threads. Disponível em https://hpc-tutorials.llnl.gov/posix/creating_and_terminating/. Acesso em 22/03/2021.

[7] GeeksforGeeks. How to Get the Id of a Current Running Thread in Java?. Disponível em https://www.geeksforgeeks.org/how-to-get-the-id-of-a-current-running-thread-in-java/. Acesso em 22/03/2021.

[8] UFSC. Slides de comunicação entre processos. Disponível em https://www.inf.ufsc.br/~lau.lung/INE5645/5.ComunicacaoentreProcessos.pdf. Acesso em 22/02/2021.

[9] GeeksforGeeks. Inter-thread Communication in Java. Disponível em https://www.geeksforgeeks.org/inter-thread-communication-java/. Acesso em 22/03/2021.

[10] Semaphores. Disponível em http://tutorials.jenkov.com/java-concurrency/semaphores.html. Acesso em 22/03/2021.