VOLTAR

Threads em JAVA e o Ambiente Multithreading

Apresentar os Threads em JAVA e um exemplo de programa Mono e Multithreading

NESTE TÓPICO

Introdução aos Conceitos sobre Processos - Revisão

O processo em sistemas operacionais, por definição, é um programa em execução na memória principal (RAM - *Randomic Access Memory*). Mas, é importante definir o que é um programa para não haver confusão no entendimento sobre o tema processos. Um programa pode ser um código-fonte que contém todos os passos para a execução de uma tarefa, ou seja, é apenas um projeto de um algoritmo específico. A outra definição: programa é um executável (binário) alocado na memória principal. Assim, quando afirmamos que o processo é um programa em execução na memória principal, estamos definindo que é um executável (programa binário) alocado em um determinado endereçamento de memória RAM.

Os processos possuem um identificador único para controle do sistema operacional e do usuário administrador denominado por *Process Identification* (ou *indentifier*) - Identificador do Processo (PID). O processo principal poderá criar outros processos durante a sua execução (estado de execução ou *running*), denominado por processos filhos, subprocessos ou threads. A Figura 1 a seguir apresenta os estados de um processo durante a sua execução.

Figura 1 - Estados ou Ciclo de Vida dos Processos

Para exemplificar o que foi discutido acima, os processos no sistema operacional GNU/Linux podem ser visualizados através do comando **top**. Veja na Figura 2 a seguir que na primeira coluna (da esquerda), está a relação de PIDs dos processos em execução na memória RAM.

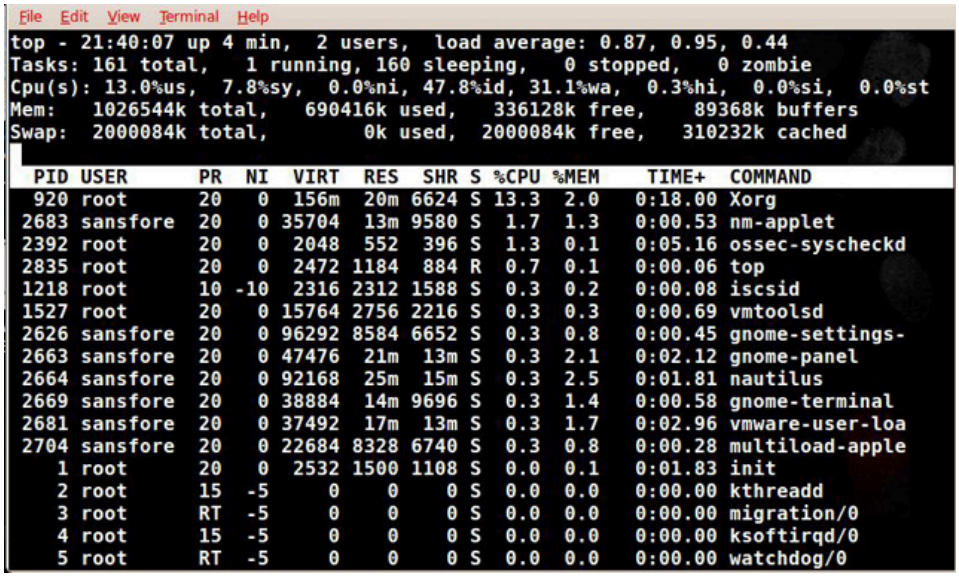

Figura 2 - Lista de processos gerada a partir do comando top do GNU/Linux

Introdução aos Conceitos sobre Threads - Revisão

Como vimos anteriomente, um processo principal poderá criar seus subprocessos ou processos filhos durante a execução e são tecnicamente conhecidos por threads. Os threads são fluxos de controle dentro do programa principal e executam tarefas específicas e normalmente pequenas. Assim, os threads são também conhecidos como processos leves ou *lightweight process*. O processo com apenas um thread é denominado por "programa monothread" e o processo com duas ou mais threads é demonomado por "programa multithreading".

A Figura 3 mostra um processo contendo um thread e outro processo contendo três threads.

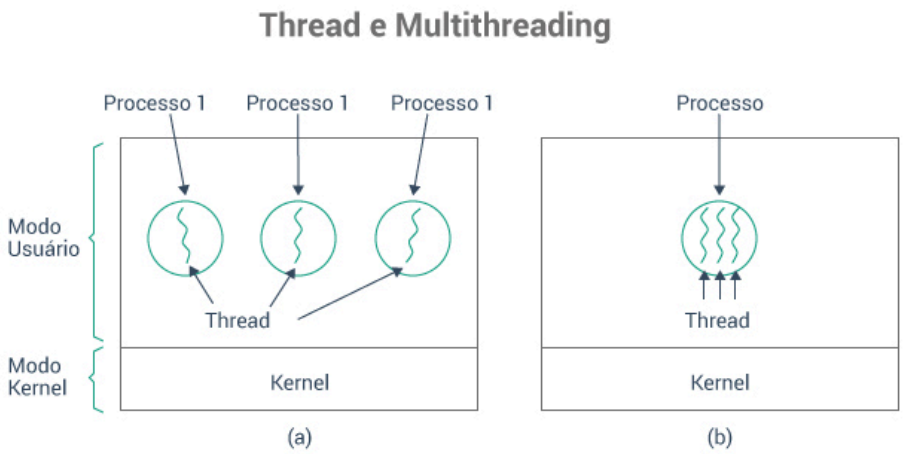

Figura 3 - Processo com uma thread e três threads.

Fonte: TANENBAUM, Andrew S.; WOODHULL, Albert S. Sistemas Operacionais: Projetjos e Implementação. Bookman, 2008.

Threads na Linguagem de Programação JAVA

Como vimos anteriormente, os threads são subprocessos que podem executar independentemente uma das outras. Assim, considerando um projeto de sistema distribuído, trabalhar com sistemas de multithreading são mais adequados. Além disso, espera-se que cada thread pode ser executado concorrentemente em um processador diferente e assim, ter uma melhoria de desempenho. Neste caso, as arquiteturas em Grade e Cluster, por exemplo, poderiam dar suporte como ambientes de processamento paralelo para sistemas multithreading.

As linguagens de programação atuais como o JAVA e o Microsoft C# (plataforma .NET), entre outras linguagens, dão suporte para a programação em multithreading. Mas, devido a portabilidade da linguagem JAVA e a sua independência no uso de threads em relação aos atuais sistemas operacionais graças as suas APIs disponíveis, adotamos a linguagem JAVA nos exemplos que apresentaremos a seguir. Em relação às linguagens mais antigas que não dão suporte ao desenvolvimento multithreading e usam as system calls (chamadas de sistema), a dependência do sistema operacional é iminente. Em outras palavras, um software que utiliza os threads desenvolvidas em C++, por exemplo, somente será executado em um determinado sistema operacional, pois o programador foi obrigado a utilizar as *system calls* daquela plataforma.

A Figura 4 a seguir mostra os estados ou ciclo de vida de um thread.

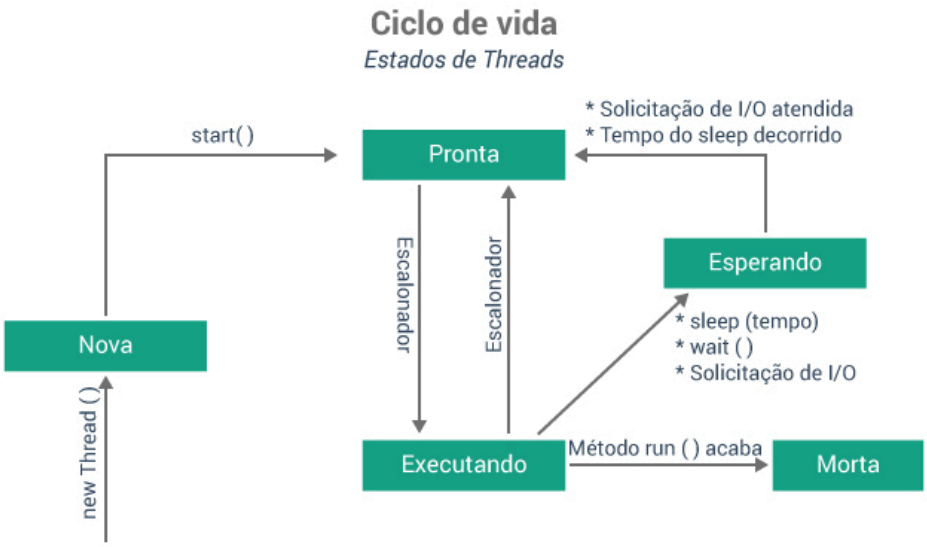

Figura 4 - O ciclo de vida (estados) de um thread

A Class Thread

Em JAVA, utilizamos a classe Thread do pacote chamado java.lang para o desenvolvimento de sistemas mono ou multithreading. A razão disso está na possibilidade de realizar execuções de cócigo paralelo e, em alguns casos, simultaneamente quando temos dois ou mais processadores disponíveis. A utilização do pacote java.lang.Thread é muito simples, pois basta fazer com que a sua classe estenda a classe Thread e implementar o método run(). Isso quer dizer que todas os seus threads executarão o código contido no método run(). O exemplo do código Java abaixo é simples, pois o método run() irá imprimir alguns dados que são passados ao método construtor MeuThread. Observe que o código abaixo é muito semelhante aos códigos apresentados anteriormente, mas com três importantes alterações, a saber:

- Na linha 1, temos a herança da Class Thread (public class MinhaThread extends java.lang.Thread) pela classe MeuThread
- Na linha 11, temos a anotação Override que é opcional e indica que o método run() será reescrito, ou seja, será executado o método run() da subclasse e não da superclasse (neste caso a Class Thread). Em outras palavras, utilizamos esta anotação quando queremos executar o método da classe filha e não método herdado pela classe mãe.
- Na linha 20, temos o método main que irá executar o programa principal
- Na linha 21, temos a instanciação do objeto denominado por thread1 e executará o método run()
- Na linha 22, temos o método start() herdada da Class Thread que despachará a thread para inicialização

Em suma, o método start() herdado da Class Thread despachará a thread1 e executará o método run() da classe MeuThread.

```
public class MeuThread extends java.lang.Thread {

        private String thread;
        private String nome;

        public MeuThread (String thread, String nome) {
              this.thread = thread;
              this.nome = nome;
        }

       @Override
       public void run(){
             System.out.println("O nome do thread é: " + nome);
             System.out.println("Thread executado: " + thread);
       }
}


public class Programa{
     public static void main (String[] args) {
              MeuThread thread1 = new MinhaThread ("Listar","Listar Produtos");
              thread1.start();
     }
}
```

Aplicativo Multithreading

O código em JAVA abaixo possui as mesmas classes, métodos, atributos que o exemplo anterior. A grande diferença é a quantidades de objetos que são instanciados e threads que são despachadas para execução. Como temos duas ou mais threads, o código demonstra a criação de um aplicativo multithreading.

```
public class MeuThread extends java.lang.Thread {

        private String thread;
        private String nome;

        public MeuThread (String thread, String nome) {
              this.thread = thread;
              this.nome = nome;
        }

       @Override
       public void run(){
             System.out.println("O nome do thread é: " + nome);
             System.out.println("Thread executado: " + thread);
       }
}


public class Programa{
     public static void main (String[] args) {
              MeuThread thread1 = new MeuThread ("Listar","Listar Clientes");
              MeuThread thread2 = new MeuThread ("Excluir","Excluir Clientes");
              MeuThread thread3 = new MeuThread ("Atualizar","Atualizar Clientes");

              thread1.start();
              thread2.start();
              thread3.start();
     }
}
```

Interface Runnable

A interface Runnable pertence ao pacote java.lang, assim como vimos anteriormente quando discutimos sobre a classe Thread. A proposta é implementar a interface Runnable que possui somente o método run(). A utilização da interface Runnable ao invés de herdar a class Thread tem algumas motivações importantes, a saber:

- A linguagem Java não permite que seja criada a herança múltipla
- A extensão a Class Thread não permite a criação de subclasses partindo de outras classes do sistema, ou seja, a criação de classes especialistas a partir de outras classes genéricas.

Partindo desta concepção, o uso da interface Runnable permite gerar classes que utilizem um thread sem a necessidade de estender a classe Thread e podendo inclusive estender outras classes genéricas. Desta forma, a geração de um thread é feita através da instanciação de um objeto thread que utilize o objeto que implementa a interface Runnable.

A interface Runnable determina como os threads vão utilizar a CPU a partir de critérios de escalonamento. Como a linguagem Java permite atribuir prioridades de execução aos threads, além do escalonamento preemptivo seguindo o conceito do algoritmo Round Robin.

O código Java abaixo, demonstra um exemplo simples que implementa a interface Runnable. Assim, vamos entender os pontos principais do código da classe MeuThreadRunnable:

- Primeiro ponto importante está na linha 3 que implementa a interface Runnable.
- As linhas 3 e 4 temos os contadores para controle.
- O laço na linha 13 irá executar n vezes cada um dos threads instanciados, ou seja, cada thread terá uma quantadade de voltas no *loop* diferente.
- Na linha 25, o método sleep() colocará os threads em estado de "dormindo" por 2 segundos.

A classe ExecutarRunnable implementa o método main que executará o programa principal. Assim, vamos entender também as linhas do código mais importantes:

- As linhas 39 a 41 estão instanciando os objetos runnable. Observe que em cada método, estamos passando como parâmetro os valores 5, 10 e 15 que configuram a quantidade de vezes que cada um dos threads passará no laço (while).
- As linhas 43, 48 e 53, temos a instanciação do thread a partir da Class Thread e passando por parâmetro o objeto runnable a ser executado.
- As linhas 44, 49 e 54 determinam os nomes dos threads instanciados.
- As linhas 46, 51 e 56 colocam os threads para inicializar (despachadas).

```
1.  package MeuThreadRunnable;
2.
3.  public class MeuRunnable implements java.lang.Runnable{
4.        private int contador;
5.      private int contador_total;
6.
7.      public MeuRunnable (int contador_total) {
8.          this.contador_total = contador_total;
9.          this.contador = 0;
10.     }
11.     public void run() {
12.
13.         while (contador <= contador_total) {
14.             System.out.println(
15.                     Thread.currentThread().getName() +
16.                     "Prioridade de Execução: " +
17.                     Thread.currentThread().getPriority() +
18.                     " - Contador atual: " + contador
19.                     );
20.             contador ++;
21.
22.             try {
23.                 System.out.println (Thread.currentThread().getName() +
24.                         " dormindo por 2 segundos...");
25.                 Thread.sleep(2000);
26.             }
27.             catch (InterruptedException e) {
28.                 e.printStackTrace(System.err);
29.             }
30.         }
31.         System.out.println ("Thread " + Thread.currentThread().getName() +
32.                         " Finalizada com Sucesso");
33.     }
34. }
35.
36. public class ExecutarRunnable {
37.
38.     public static void main(String[] args) {
39.         MeuRunnable runnable1 = new MeuRunnable (5);
40.         MeuRunnable runnable2 = new MeuRunnable (10);
41.         MeuRunnable runnable3 = new MeuRunnable (15);
42.
43.         Thread thread1 = new Thread(runnable1);
44.         thread1.setName("Runnable 1 - ");
45.
46.         thread1.start();
47.
48.         Thread thread2 = new Thread(runnable2);
49.         thread2.setName("Runnable 2 - ");
50.
51.         thread2.start();
52.
53.         Thread thread3 = new Thread(runnable3);
54.         thread3.setName("Runnable 3 - ");
55.
56.         thread3.start();
57.
58.         System.out.println ("Threads Runnable Inicializados com Sucesso.");
59.     }
60.  }
```

Quiz

Exercício

Threads em JAVA e o Ambiente Multithreading

INICIAR

Referências

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML - Guia do Usuário: Tradução da Segunda Edição**. Elsevier Brasil, 2000.

TANENBAUM, Andrew S.; WOODHULL, Albert S. **Sistemas Operacionais: Projetjos e Implementação**. Bookman, 2008.

LEA, Douglas. **Concurrent programming in Java: design principles and patterns**. Addison-Wesley Professional, 2000.

Avalie este tópico

ANTERIOR
Threads em JAVA e o Ambiente Multithreading

Índice

PRÓXIMO
Problemas de Concorrência e

Ajuda?
(https://ava.un
idCurso=)

Biblioteca (https://www.uninove.br/conheca-a-uninove/biblioteca/sobre-a-biblioteca/apresentacao/)
Portal Uninove (http://www.uninove.br)
Mapa do Site

Compartilham