Computação Distribuída

Concorrência 1

António Rui Borges

Sumário

- *Programa vs. Processo*
  - *Caracterização de um ambiente multiprogramado*
- *Processos vs. Threads*
  - *Caracterização de um ambiente multithreaded*
- *Threads em Java*
- *Leituras sugeridas*

Programa vs. Processo

De um modo geral, um *programa* pode ser definido como um conjunto de instruções que descreve a realização de uma determinada tarefa por um computador. Contudo, para que essa tarefa seja *de facto* realizada, o programa correspondente tem que ser executado.

A execução de um programa designa-se de *processo*.

Tratando-se da representação de uma actividade em curso, o *processo* caracteriza-se em cada instante pelo(s)

- código e valor actual de todas as variáveis associadas (*espaço de endereçamento*);
- valor actual de todos os registos internos do processador;
- dados que estão a ser transferidos dos dispositivos de entrada e para os dispositivos de saída;
- estado de execução.

Modelação dos processos - 1

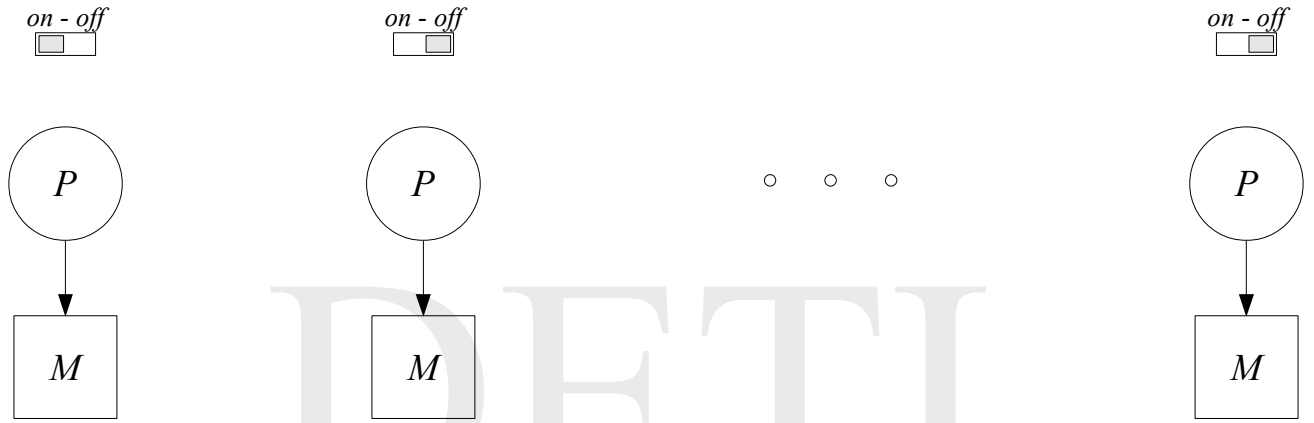
A *multiprogramação*, ao criar uma imagem de aparente simultaneidade na execução de diferentes programas pelo mesmo processador, torna muito complexa a percepção das diferentes actividades que estão em curso.

Esta imagem pode, porém, ser simplificada se, em vez de se procurar seguir o percurso do processador nas suas constantes comutações entre processos, se supuser a existência de um conjunto de processadores virtuais, um por cada processo que concorrentemente coexiste, e se admitir que os processos associados são executados em paralelo através da activação (*on*) e da desactivação (*off*) dos processadores respectivos.

Para que tal modelo seja viável, é preciso garantir que

- a execução dos processos não é afectada pelo instante, ou local no código, onde ocorre a comutação;
- não são impostas quaisquer restrições relativamente aos tempos de execução, totais ou parciais, dos processos.

Modelação dos processos - 2



- a *comutação de contexto* é simulada pela activação e desactivação dos processadores virtuais e é controlada pelo seu *estado*;
- num *monoprocessador*, o número de processadores virtuais activos em cada instante é no máximo um;
- num *multiprocessador* o número de processadores virtuais activos em cada instante é no máximo igual ao números de processadores existentes.

Diagrama de estados de um processo - 1

Um processo vai encontrar-se em diferentes situações, designadas de *estados*, ao longo da sua existência.

Os estados mais importantes são os seguintes

- *run* – quando detém a posse do processador e está, por isso, em execução;
- *ready-to-run* – quando aguarda a atribuição do processador para começar ou continuar a sua execução;
- *blocked* – quando está impedido de continuar até que um acontecimento externo ocorra (acesso a um recurso, completamento de uma operação de entrada / saída, etc.).

As transições entre estados resultam normalmente de uma intervenção externa, mas podem nalguns casos ser despoletadas pelo próprio processo.

A parte do sistema de operação que lida com estas transições designa-se *scheduler* [neste caso, *do processador*] e constitui parte integrante do seu núcleo central (o *kernel*) que é responsável pelo tratamento das interrupções e por agendar a atribuição do processador e dos restantes recursos do sistema computacional.

Diagrama de estados de um processo - 2

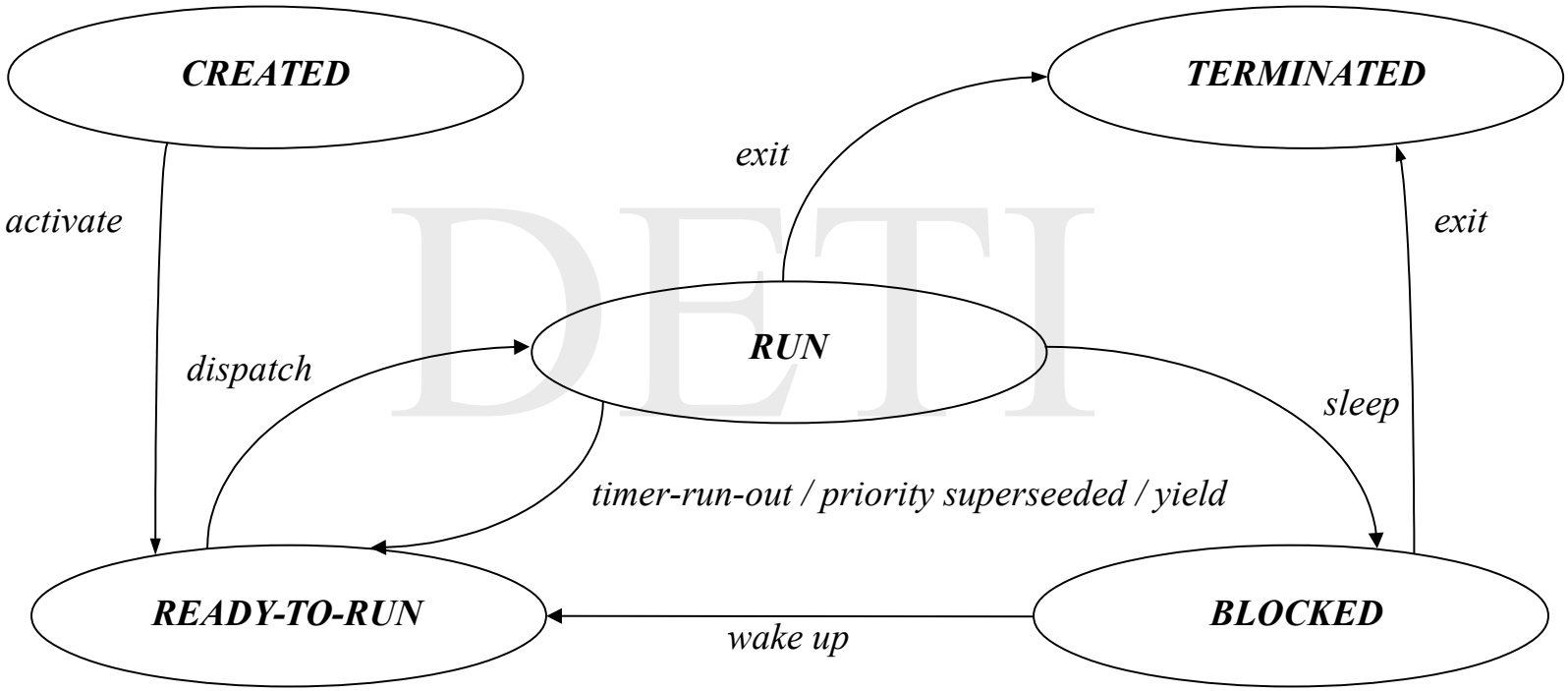


Diagrama de estados de um processo - 3

activate – o processo é criado e colocado na *fila de espera dos processos prontos a serem executados* aguardando ser calendarizado para execução;

dispatch – um dos processos da *fila de espera dos processos prontos a serem executados* é seleccionado pelo *scheduler* para execução;

timer-run-out – o processo em execução esgotou o intervalo de tempo de processador que lhe tinha sido atribuído (*preemptive scheduling*);

priority superseded – o processo em execução perde o processador porque surgiu entretanto na *fila de espera dos processos prontos a serem executados* um processo de prioridade mais elevada (*preemptive scheduling*);

yield – o processo prescinde voluntariamente do processador para permitir a execução de outros processos (*non-preemptive scheduling*);

sleep – o processo está impedido de prosseguir, aguardando a ocorrência de um acontecimento externo;

wake up – ocorreu entretanto o acontecimento externo que o processo aguardava;

exit – o processo termina a sua execução e aguarda a libertação dos recursos que lhe estavam atribuídos.

Processos vs. Threads - 1

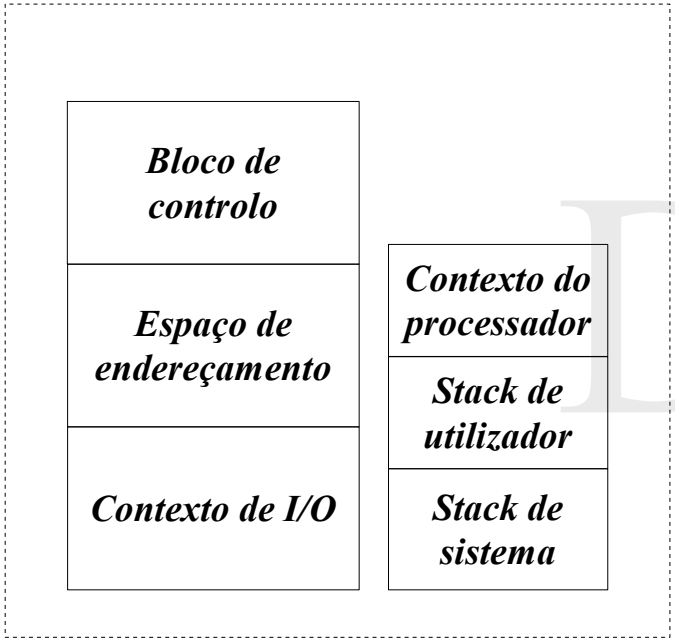
O conceito de *processo* corporiza as propriedades seguintes

- *pertença de recursos* – um espaço de endereçamento próprio e um conjunto de canais de comunicação com os dispositivos de entrada / saída;
- *fio de execução* (*thread*) – um *program counter* que sinaliza a localização da instrução que deve ser executada a seguir, um conjunto de *registos internos do processador* que contêm os valores actuais das variáveis em processamento e um *stack* que armazena a história de execução (um *frame* por cada rotina invocada e que ainda não retornou).

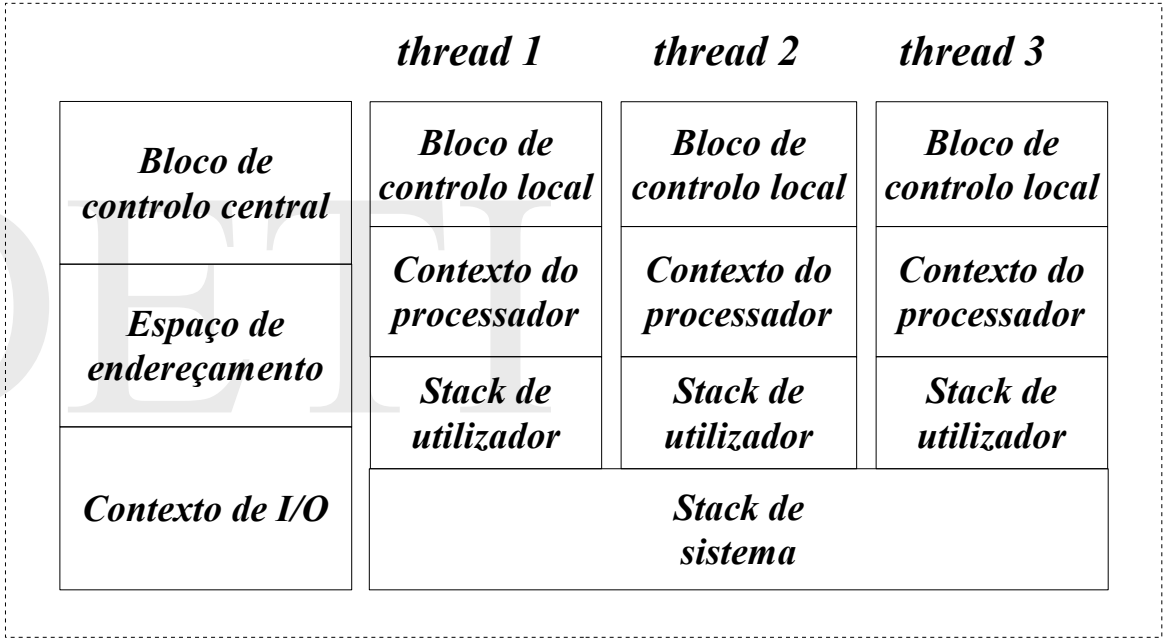
Estas propriedades, embora surjam reunidas num *processo*, podem ser tratadas separadamente pelo ambiente de execução. Quando tal acontece, os *processos* dedicam-se a agrupar um conjunto de recursos e os *threads*, também conhecidos por *light weight processes*, constituem entidades executáveis independentes dentro do contexto de um mesmo processo.

Multithreading representa então a situação em que é possível criar-se *threads* de execução múltiplos no contexto de um processo.

Processos vs. Threads - 2



Single threading

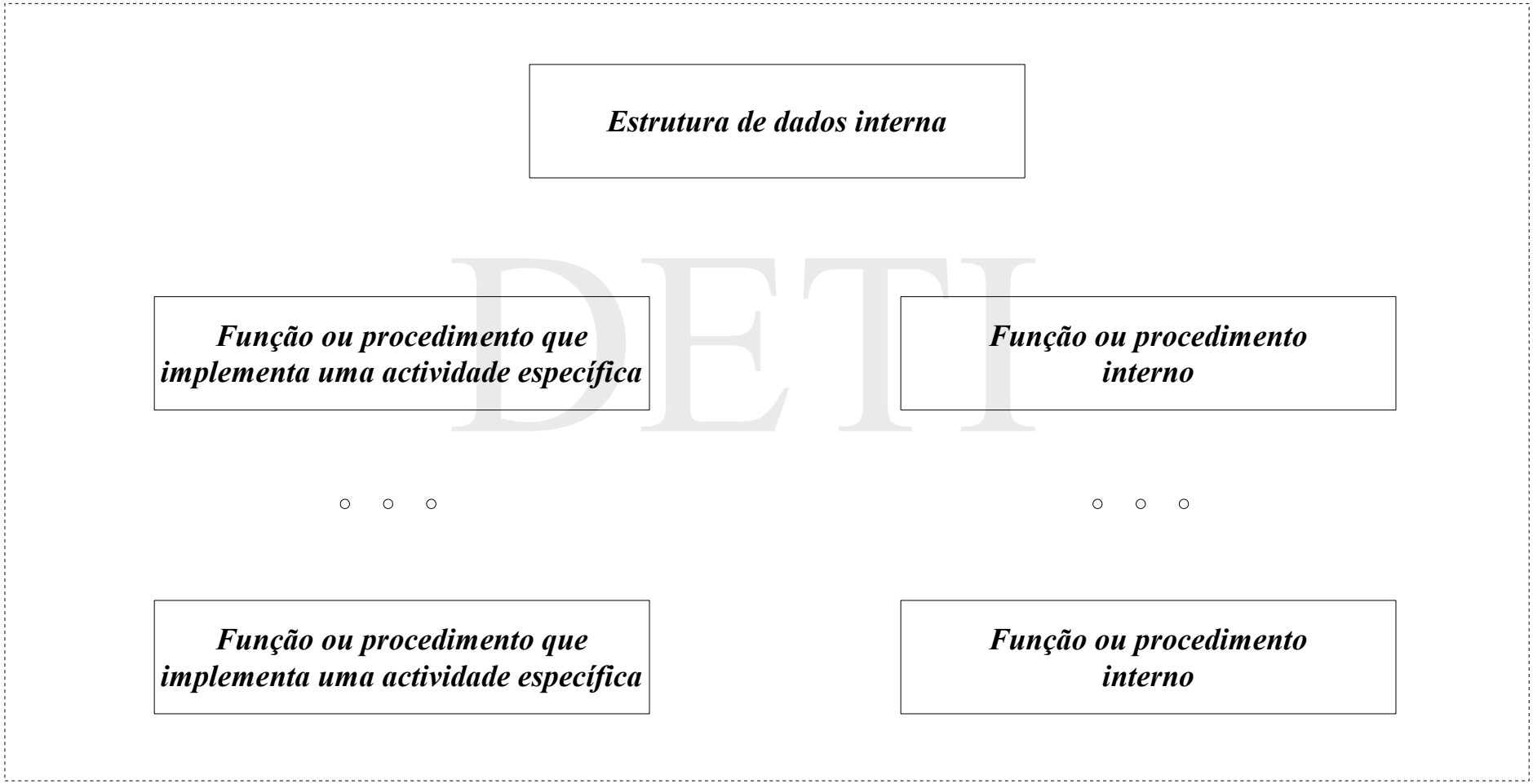


Multithreading

Vantagens de um ambiente multithreaded

- *maior simplicidade na decomposição da solução e maior modularidade na implementação* – programas que envolvem múltiplas actividades e atendem múltiplas solicitações são mais fáceis de conceber e de implementar numa perspectiva concorrencial do que numa perspectiva puramente sequencial
- *melhor gestão de recursos do sistema computacional* – havendo uma partilha do espaço de endereçamento e do contexto de *I/O* entre os *threads* que compõem uma aplicação, torna-se mais simples gerir a ocupação da memória principal e o acesso eficiente aos dispositivos de entrada / saída
- *eficiência e velocidade de execução* – uma decomposição da solução em *threads* por oposição a processos, ao envolver menos recursos por parte do sistema de operação, possibilita que operações como a sua criação e destruição e a mudança de contexto se tornem menos pesadas e, portanto, mais eficientes; além disso, em multiprocessadores simétricos torna-se possível calendarizar para execução em paralelo múltiplos *threads* da mesma aplicação, aumentando assim a velocidade de execução.

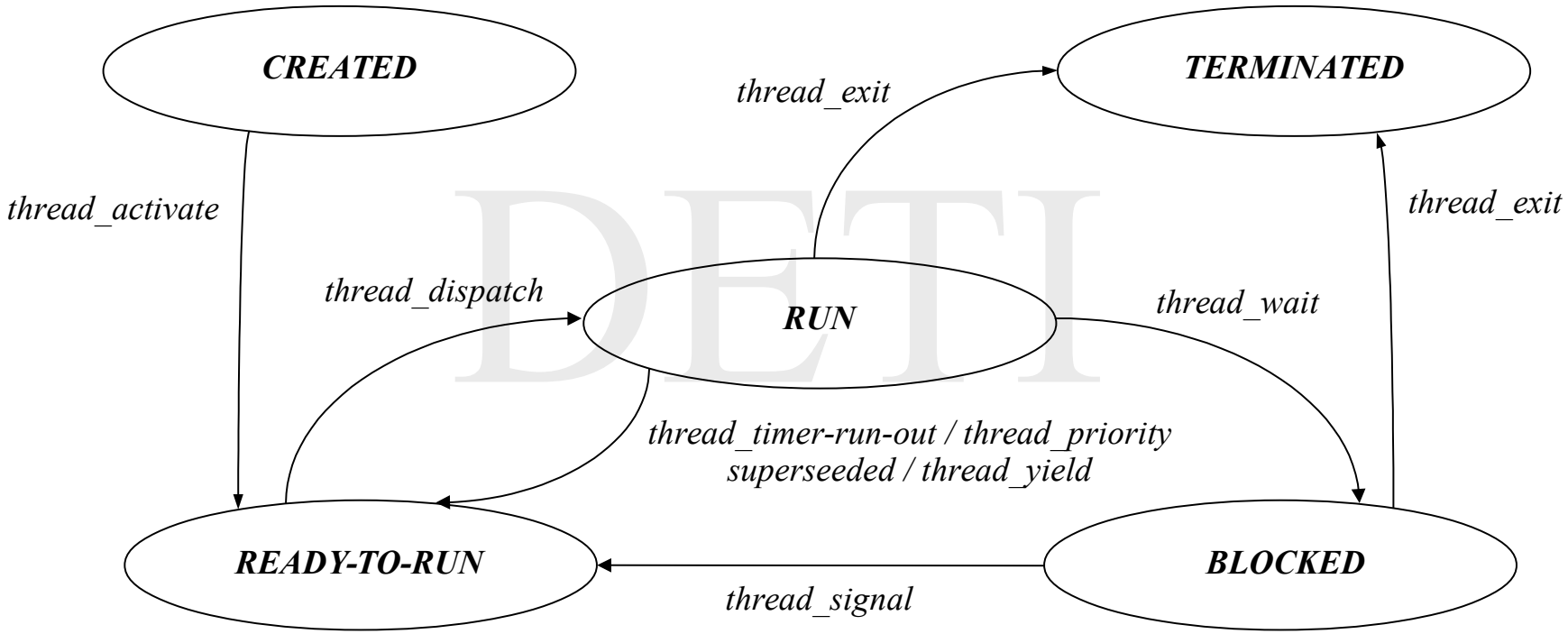
Organização de um programa multithreaded - 1



Organização de um programa multithreaded - 2

- cada *thread* está tipicamente associado à execução de uma *função ou procedimento que implementa uma actividade específica*;
- a comunicação entre os múltiplos *threads* que coexistem num dado instante é materializada pelo acesso à *estrutura de dados interna*, que é global, e onde está definido um espaço de partilha de informação em termos de variáveis e de canais de comunicação com os dispositivos de entrada / saída;
- o *programa principal*, representado no diagrama por uma *função ou procedimento que implementa uma actividade específica*, constitui o primeiro *thread* a ser criado e, tipicamente, o último *thread* a ser concluído.

Diagrama de estados de um thread



Suporte à implementação de um ambiente multithreaded

- *user level threads* – os *threads* são implementados por uma biblioteca específica ao nível utilizador que fornece apoio à criação, gestão e *scheduling* de *threads* sem interferência do *kernel*; isto significa que a implementação é muito versátil e portável, mas, como o *kernel* vê apenas o processo a que eles pertencem, quando um *thread* particular executa uma *chamada ao sistema* bloqueante, todo o processo é bloqueado, mesmo que existam *threads* que estejam prontos a serem executados
- *kernel level threads* – os *threads* são implementados directamente ao nível do *kernel* que providencia as operações de criação, gestão e *scheduling* de *threads*; a sua implementação é menos versátil do que no caso anterior, mas o bloqueio de um *thread* particular não afecta a calendarização para execução dos restantes e torna-se possível a sua execução paralela num multiprocessador.

Threads em Java - 1

Sendo Java uma linguagem concorrente, os *threads* são suportados ao nível da própria linguagem. Conceptualmente, o lançamento de um *thread* em Java pressupõe a existência de duas entidades: um objecto que representa um fio de execução autónomo em termos da máquina virtual, o *thread* propriamente dito, e um tipo de dados não instanciado, ou um objecto instanciado a partir dele, que contém o método executado pelo *thread*.

A *máquina virtual de Java* gere o ambiente *multithreading* de acordo com as regras seguintes

- todo o programa supõe pelo menos um *thread* que é implicitamente lançado quando a máquina virtual, após instalar o ambiente de execução, invoca o método `main` do tipo de dados de arranque;
- os *threads* restantes são explicitamente criados e lançados pelo *thread* `main` ou por um *thread* que resultou de lançamento em sucessão a partir do *thread* `main`;
- o programa termina quando todos os *threads* tiverem terminado a execução do método que lhes está associado.

Threads em Java - 2

A biblioteca base de Java, `java.lang`, fornece dois tipos de dados, um usando o construtor *interface* e outro usando o construtor *class*, que são instrumentais na construção do ambiente *multithreading*.

```
public interface Runnable
{
  public void run ();
}
```

```
public class Thread
{
    . . .
  public void run ()
  public void start ()
    . . .
}
```

Threads em Java - 3

Cada fio de execução autónomo é enquadrado pela máquina virtual de Java como uma instanciação do tipo de dados `Thread`. Estão nele definidos dois métodos que são operacionalmente importantes neste contexto

- o método `run` – que é invocado quando o *thread* é posto em execução (lançado);
- o método `start` – que é invocado para lançar o *thread*.

Não é, porém, estritamente necessário criar novos tipos de dados, derivados do tipo `Thread` e que façam o *overriding* do método `run`, para se garantir a execução de tarefas específicas.

O mesmo pode ser conseguido, de um modo alternativo, criando um tipo de dados independente que implemente o interface `Runnable` e que contenha uma implementação do método `run`.

Threads em Java - 4

Lançamento de um thread (método 1)

instanciação

```
. . .
MyThread thr = new MyThread ();

thr.start ();
. . .
```

lançamento

overriding do método `run` *que estabelece a operacionalidade do thread*

```
public class MyThread extends Thread
{
        . . .
  public void run ()
  {
       . . .
  }
}
```

tipo de dados que define a funcionalidade do thread

Threads em Java - 5

Lançamento de um thread (método 2)

instanciação

```
. . .
Thread thr = new Thread (new MyThread ());

thr.start ();
. . .
```

lançamento

implementação do método `run` *que estabelece a operacionalidade do thread*

```
public class MyThread implements Runnable
{
        . . .
  public void run ()
  {
      . . .
  }
}
```

tipo de dados que define a funcionalidade do thread

Threads em Java - 6

Um *thread* apresenta os atributos seguintes

- *nome* – nome atribuído (por defeito, a máquina virtual de Java gera um *string* de formato `Thread-#`, em que `#` representa o número de lançamento que é sucessivamente incrementado a partir de zero);
- *identificador interno* – valor de tipo `long` que é único e se mantém inalterado durante o tempo de vida do *thread*;
- *grupo* – grupo a que o *thread* pertence (todos os *threads* de uma aplicação pertencem por defeito ao mesmo grupo, o grupo `main`);
- *prioridade* – pode variar entre 1 (`MIN_PRIORITY`) e 10 (`MAX_PRIORITY`), por defeito, a máquina virtual de Java atribui a prioridade 5 (`NORM_PRIORITY`);
- *estado* – o estado actual do *thread*
  - `NEW` (*CREATED*), após a instanciação da variável correspondente de tipo `Thread`, ou de um tipo derivado deste
  - `RUNNABLE` (*READY-TO-RUN* ou *RUN*), quando está em execução ou a aguardar execução
  - `BLOCKED`, `WAITING` ou `TIME_WAITING` (*BLOCKED*), quando está bloqueado
  - `TERMINATED` (*TERMINATED*), após a sua terminação.

Informação disponível sobre um thread

Thread characterization

```
Name = main
Internal identifier = 1
Group = main
Priority = 5
Current state = RUNNABLE

Possible states:
   NEW
   RUNNABLE
   BLOCKED
   WAITING
   TIMED_WAITING
   TERMINATED
```

Threads em Java - 7

Uma aplicação *multithreaded* em Java termina em princípio quando todos os *threads* constituintes terminarem. Em aplicações complexas, com um número de *threads* de suporte muito elevado, lidar com situações excepcionais que conduzam à necessidade de terminação de operações, pode assim tornar-se muito complicado e exigir a introdução de código específico cuja utilidade prática é questionável.

Para obviar o problema, o Java apresenta duas alternativas

- *invocação do método* `void System.exit (int status)` – que conduz à terminação forçada da máquina virtual de Java em execução, fornecendo o *status* de operação comunicado;
- transformar os *threads* instanciados, directa ou indirectamente, a partir do *thread* `main` em *daemons* – a máquina virtual de Java termina logo que todos os *threads* remanescentes têm esta propriedade.

Threads em Java - 8

No entanto, a terminação habitual de uma aplicação *multithreaded* é efectuada colocando o *thread* principal a aguardar a terminação de todos os *threads* que tenham sido eventualmente lançados a partir dele.

Em Java, a situação é semelhante. O tipo de dados `Thread` contém o método `join` que, como é tradicional e numa perspectiva orientada por objectos, bloqueia o *thread* invocador até que o *thread* referenciado termine.

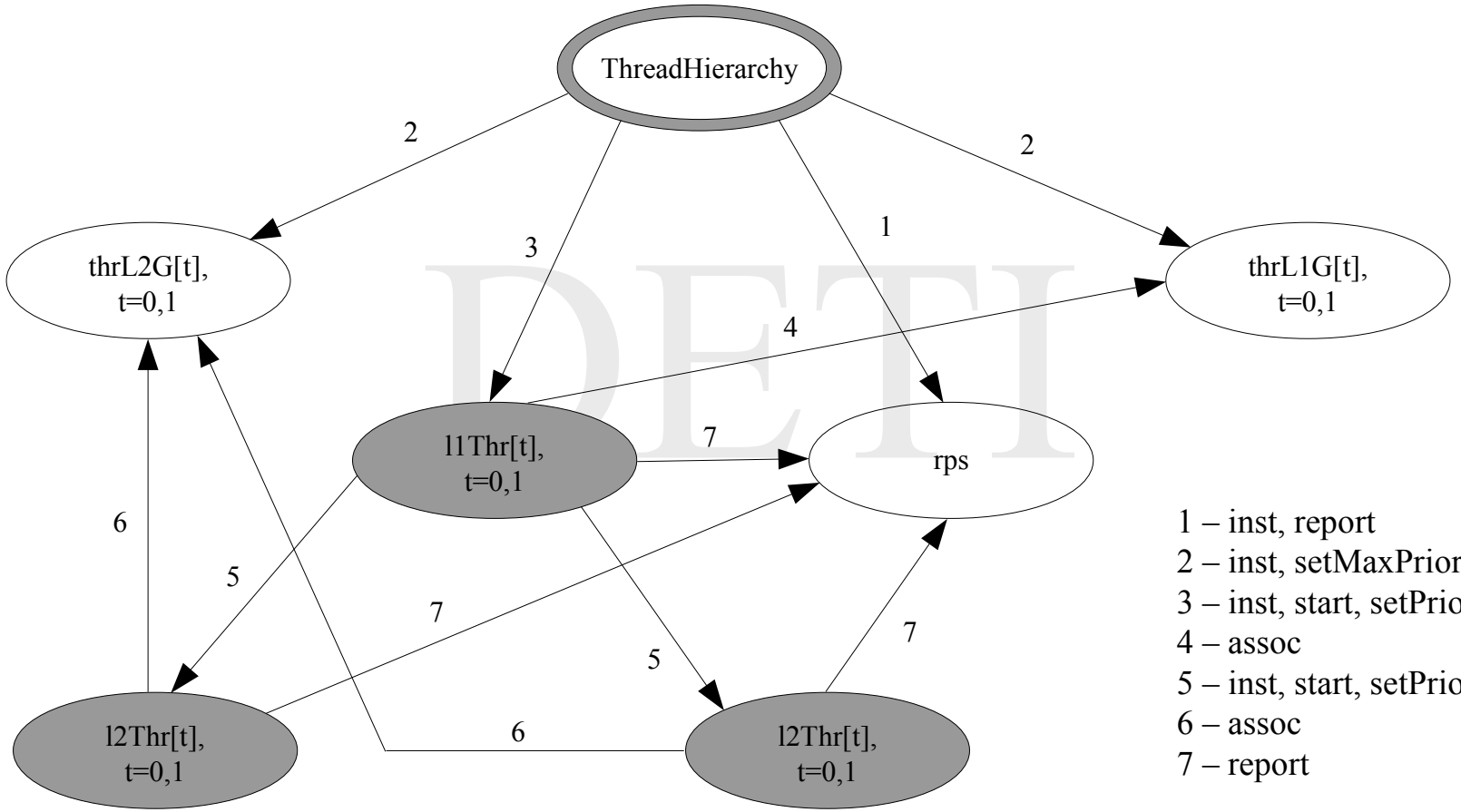
Threads em Java - 9

A biblioteca base de Java, `java.lang`, fornece um tipo de dados `ThreadGroup` para organização dos *threads* em grupos. A compartimentação em grupos diferenciados dos *threads* sucessivamente instanciados possibilita não só organizar hierárquica e funcionalmente uma aplicação, como também tirar partido de particularidades do Java na definição de propriedades comuns e na interacção entre eles.

Tem-se assim que

- é possível fixar à partida a prioridade máxima e a propriedade de ser *daemon* associadas a um dado grupo de *threads* – todos os *threads* posteriormente instanciados, e pertencentes a esse grupo, manterão estas características;
- é possível enviar uma interrupção a *todos* os *threads* pertencentes a um grupo numa operação única, em vez de fazê-lo separadamente a cada um dos elementos do grupo.

Criação e lançamento hierárquicos de threads - 1



Criação e lançamento hierárquicos de threads - 2

nível 0

threadName: main
threadId: 1
threadPriority: 5
threadGroupName: main
threadParentGourpName: system

nível 1

threadName: Thread_L1.1
threadId: 8
threadPriority: 9
threadGroupName: Thread_G1.1
threadParentGroupName: main

threadName: Thread_L1.2
threadId: 9
threadPriority: 8
threadGroupName: Thread_G1.2
threadParentGroupName: main

threadName: Thread_L1.1_L2.1
threadId: 11
threadPriority: 8
threadGrpName: Thread_G1.1_G2
threadPrtGrpName: Thread_G1.1

threadName: Thread_L1.1_L2.2
threadId: 12
threadPriority: 8
threadGrpName: Thread_G1.1_G2
threadPrtGrpName: Thread_G1.1

threadName: Thread_L1.2_L2.1
threadId: 13
threadPriority: 7
threadGrpName: Thread_G1.2_G2
threadPGroupName: Thread_G1.2

threadName: Thread_L1.2_L2.2
threadId: 14
threadPriority: 7
threadGrpName: Thread_G1.2_G2
threadPGroupName: Thread_G1.2

nível 2

Criação e lançamento hierárquicos de threads - 3

Valores impressos

```
Number of level 1 threads? 2
Number of level 2 threads per level 1 threads? 2

Thread name: main
Thread id: 1
Thread priority: 5
Name of the thread group: main
Name of the parent group of the thread group: system
N. of active threads in the thread group: 1
Name of active threads in the thread group: main
N. of active subgroups in the thread group: 0

Thread name: Thread_L1.1_L2.1
Thread id: 11
Thread priority: 8
Name of the thread group: Thread_G1.1_G2
Name of the parent group of the thread group: Thread_G1.1
N. of active threads in the thread group: 2
Name of active threads in the thread group: Thread_L1.1_L2.1 -
N. of active subgroups in the thread group: 0
                                        .   .   .
```

Threads em Java - 10

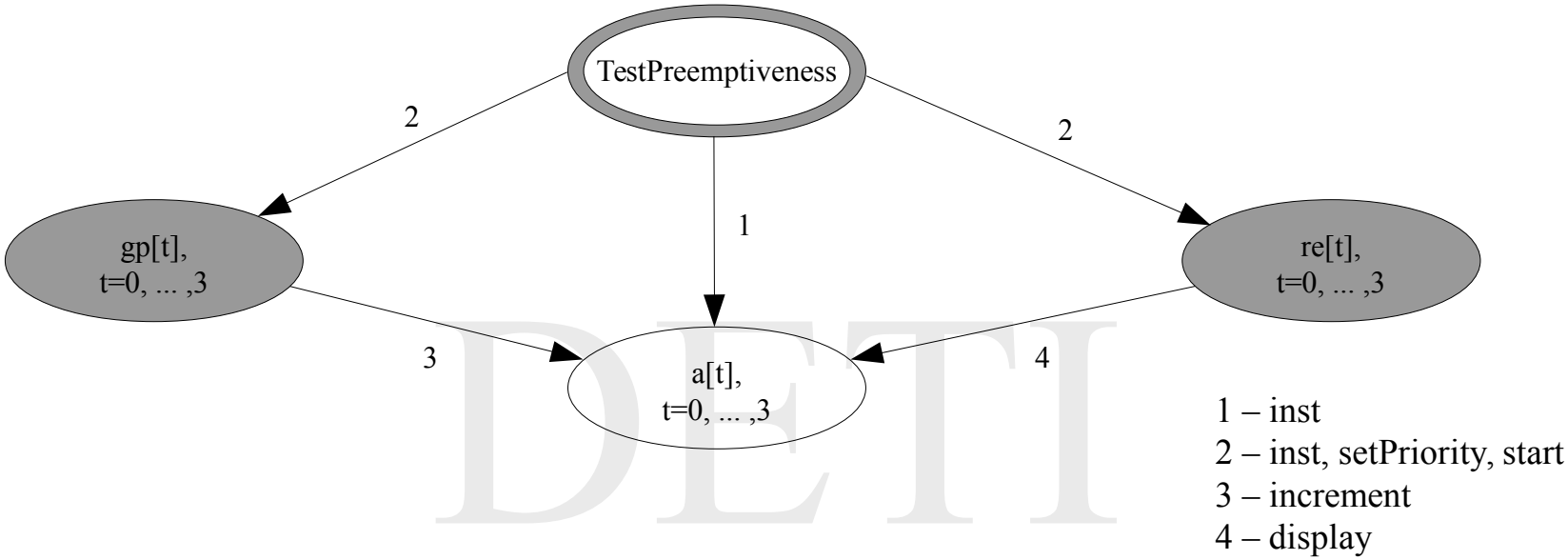
A máquina virtual de Java supõe uma política de *scheduling non-preemptive* baseada num sistema de prioridades estáticas de 10 níveis

- as transições entre o estado *RUN* e o estado *READY-TO-RUN* são de tipo *thread_priority_superseded* e *thread_yield*;
- a prioridade atribuída a um *thread*, estabelecida quando ele é instanciado ou modificada antes do seu lançamento, permanece inalterada durante o seu tempo de vida activa.

A máquina virtual de Java não impõe, contudo, a política de *scheduling* de um modo estrito. É deixada à implementação bastante liberdade na forma como ela é operacionalizada. Isto é particularmente verdadeiro quando a máquina virtual de Java é executada sobre um sistema de operação interactivo de uso geral!

Em Linux, por exemplo, os *threads* de Java são implementados ao nível do *kernel*, tirando partido de plataformas *multicore* para potenciar a sua execução paralela, e a política de *scheduling* prevalecente é a política local.

Impacto da definição de prioridade - 1



São criados dois tipos de *threads*

- *threads* computacionalmente intensivos, `gp[.]`, cuja função é incrementar uma variável inteira `a[.]` dez milhões de vezes
- *threads* I/O intensivos, re`[.]`, cuja função é ler o valor actual da variável `a[.]`.

A prioridade de cada tipo pode ser fixada.

Impacto da definição de prioridade - 2

Sem yield do processador

```
Priority level of computation intensive threads? 1
Priority level of I/O intensive threads? 10
Priority of computation intensive threads = 1
Priority of I/O intensive threads = 10
I have already started the I/O intensive threads!
I have already started the computation intensive threads!
I have finished my job!
C = 10000000
N. of iterations in printing values of C = 227407
A = 10000000
N. of iterations in printing values of A = 256322
B = 10000000
D = 10000000
N. of iterations in printing values of B = 248244
N. of iterations in printing values of D = 227383
```

Impacto da definição de prioridade - 3

Sem yield do processador

```
Priority level of computation intensive threads? 10
Priority level of I/O intensive threads? 1
Priority of computation intensive threads = 10
Priority of I/O intensive threads = 1
I have already started the I/O intensive threads!
I have already started the computation intensive threads!
I have finished my job!
B = 10000000
N. of iterations in printing values of B = 137225
A = 10000000
N. of iterations in printing values of A = 153943
D = 10000000
N. of iterations in printing values of D = 143686
C = 10000000
N. of iterations in printing values of C = 168529
```

Impacto da definição de prioridade - 4

Com yield do processador

```
Priority level of computation intensive threads? 1
Priority level of I/O intensive threads? 10
Priority of computation intensive threads = 1
Priority of I/O intensive threads = 10
I have already started the I/O intensive threads!
I have already started the computation intensive threads!
I have finished my job!
N. of iterations in printing values of A = 6768171
A = 10000000
C = 10000000
N. of iterations in printing values of C = 6898383
D = 10000000
N. of iterations in printing values of D = 7049625
B = 10000000
N. of iterations in printing values of B = 6982515
```

Impacto da definição de prioridade - 5

Com yield do processador

```
Priority level of computation intensive threads? 10
Priority level of I/O intensive threads? 1
Priority of computation intensive threads = 10
Priority of I/O intensive threads = 1
I have already started the I/O intensive threads!
I have already started the computation intensive threads!
I have finished my job!
A = 10000000
N. of iterations in printing values of A = 6968156
C = 10000000
N. of iterations in printing values of C = 6867627
B = 10000000
N. of iterations in printing values of B = 6820398
D = 10000000
N. of iterations in printing values of D = 6850112
```

Threads em Java - 11

O campo `a` do tipo de dados `Variable` inclui o modificador `volatile`. O seu significado preciso é informar o compilador de Java que todos os *threads*, durante a sua execução, têm que observar em permanência um valor *consistente* da variável `a`.

Consistência, neste contexto, quer dizer que o acesso à variável `a` se processa sempre exactamente da maneira prescrita pelo código de cada *thread*.

Esta informação é no caso presente fundamental porque o modelo de memória de Java permite que, quer o compilador, ao gerar o *bytecode* correspondente a um dado tipo de dados, quer a máquina virtual, ao interpretar esse *bytecode*, possam realizar optimizações ao código original que, sendo inteiramente consistentes num ambiente *singlethreaded*, podem originar execuções aparentemente paradoxais num ambiente *multithreaded*.

Threads em Java – 12

O ambiente de execução fornecido pela máquina virtual de Java (JVM) pode ser acedido através do método `getRuntime()` do tipo de dados `Runtime` da biblioteca base de Java, `java.lang`. A partir dele, torna-se possível obter informação sobre o número de processadores actualmente acessíveis à JVM, a memória disponível, a introdução de mecanismos para o *tracing* de instruções e/ou de invocação de métodos e o lançamento de processos autónomos ao nível do sistema de operação subjacente..

Adicionalmente, através dos métodos `getProperties()` e `getEnv()` do tipo de dados `System` da biblioteca base de Java, `java.lang`, é possível obter-se as propriedades do ambiente e as variáveis do ambiente de interacção com o utilizador fornecidas pelo sistema de operação subjacente, por exemplo, a *shell* `bash` em Linux.

Informação sobre o ambiente de execução - 1

Characterization of Java Virtual Machine (JVM)

```
N. of available processors = 2
Size of dynamic memory presently free (in bytes) = 47227584
Size of total dynamic memory (in bytes) = 48234496
Maximum size of available main memory of the hardware platform where Java
virtual machine is installed (in bytes) = 703070208
```

Properties of the execution environment

```
java.runtime.name = Java(TM) SE Runtime Environment
sun.boot.library.path = /opt/jdk1.8.0_152/jre/lib/amd64
path.separator = :
java.vm.name = Java HotSpot(TM) 64-Bit Server VM
java.vm.specification.name = Java Virtual Machine Specification
user.dir =
  /home/ruib/Programs/java/NetBeansProjects/CDExamples/basicThreads/environment
java.runtime.version = 1.8.0_152-b16
os.name = Linux
sun.jnu.encoding = UTF-8
user.home = /home/ruib
file.encoding = UTF-8
user.name = ruib
sun.arch.data.model = 64
java.home = /opt/jdk1.8.0_152/jre
                                        .   .   .
```

Informação sobre o ambiente de execução - 2

Variables of the execution environment

```
PATH =
  /opt/jdk1.8.0_152/jre/bin:/opt/jdk1.8.0_152/bin:/usr/lib64/qt-3.3/bin:
  /usr/lib64/ccache:/usr/local/bin:/usr/bin:/bin:/usr/games:/usr/local/sbin:
  /usr/sbin:/home/ruib/.local/bin:/home/ruib/bin
LD_LIBRARY_PATH =
  /opt/jdk1.8.0_152/jre/lib/amd64:/opt/jdk1.8.0_152/jre/lib/i386:
SESSION_MANAGER = local/unix:@/tmp/.ICE-unix/1337,unix/unix:/tmp/.ICE-unix/1337
LOGNAME = ruib
PWD = /home/ruib
SSH_ASKPASS = /usr/bin/ksshaskpass
LANG = en_US.UTF-8
XDG_SESSION_ID = 1
NB_DESKTOP_STARTUP_ID =
  ruib-laptop3.ieeta.pt;1519649015;947808;1466_TIME4217848
DISPLAY = :0
USER = ruib
HOSTNAME = ruib-laptop3.ieeta.pt
HOME = /home/ruib
                                    .    .    .
```

Threads em Java – 13

O tipo de dados `ProcessBuilder` da biblioteca base de Java, `java.lang`, possibilita a criação de processos do sistema de operação subjacente no contexto da máquina virtual de Java através do método start `()`. Estes processos são vistos como instanciações do tipo de dados `Process` da mesma biblioteca.

Neste âmbito, os atributos seguintes do processo podem ser inicializados

- um *comando*, formado por uma lista de *strings* que descrevem o caminho para o ficheiro a ser executado e respectivos parâmetros
- um *ambiente de execução*
- um *directório de trabalho*
- uma fonte para o *stream* de entrada standard
- destinos para os *streams* de saída standard e de erro.

Execução de um comando ao nível do sistema de operação

Listing current working directory

```
----------------------------------------
total 28
drwxrwxr-x. 5 ruib ruib 4096 Feb 25 11:25 .
drwxrwxr-x. 8 ruib ruib 4096 Feb 26 18:14 ..
drwxrwxr-x. 3 ruib ruib 4096 Feb 25 11:25 build
-rw-rw-r--. 1 ruib ruib 3542 Feb 25 11:24 build.xml
-rw-rw-r--. 1 ruib ruib   82 Feb 25 11:24 manifest.mf
drwxrwxr-x. 3 ruib ruib 4096 Feb 25 11:24 nbproject
drwxrwxr-x. 2 ruib ruib 4096 Feb 25 11:25 src
----------------------------------------
exit status = 0
```

Leituras sugeridas

- *Distributed Systems: Concepts and Design, 4th Edition, Coulouris*, Dollimore, Kindberg, Addison-Wesley
  - Capítulo 6: *Operating systems support*
- *Distributed Systems: Principles and Paradigms, 2nd Edition,* Tanenbaum, van Steen, Pearson Education Inc.
  - Capítulo 3: *Processes*