Unidade 2

Programação Paralela

- Processos
- Threads
- Paralelismo em Java

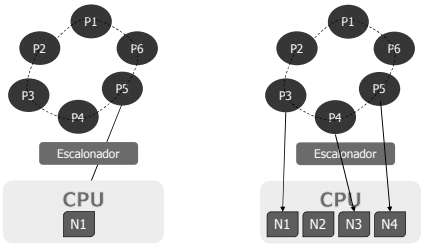
Processos

- Definição
  - Um programa em execução em uma máquina
  - Identificado pelo seu PID (*Process Identifier*)
- Execução dos Processos
  - Um processador pode executar somente um processo a cada instante
  - Em um S.O. multi-tarefa, processos se alternam no uso do processador – cada processo é executado durante um *quantum* de tempo
  - Se houver N processadores, N processos podem ser executados simultaneamente

2

Processos

- Escalonamento: 1 núcleo x n núcleos



3

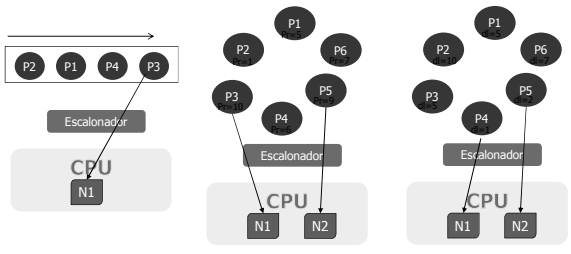
Processos

- Escalonamento de Processos
  - O escalonador do S.O. seleciona o(s) processo(s) que deve(m) ser executado(s) pelo(s) processador(es)
- Algoritmo de Escalonamento
  - Define a ordem de execução de processos com base em uma fila, prioridade, deadline, ...
  - Em geral, os sistemas adotam uma política de melhor esforço para atender a todos os processos de maneira justa e igualitária
  - Processos do sistema e aplicações críticas (um alarme, por exemplo) exigem maior prioridade

4

Processos

- Escalonamento: 1 núcleo x n núcleos



5

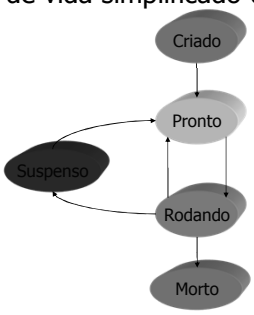
Processos

- Estados de um Processo
  - Pronto: processo pronto para ser executado, mas sem o direito de usar o processador
  - Rodando: sendo executado pelo processador
  - Suspenso: aguarda operação de I/O, liberação de um recurso ou fim de tempo de espera
- Processos trocam de estado de acordo com:
  - Algoritmo de escalonamento
  - Troca de mensagens
  - Interrupções de hardware ou software

6

Processos

- Ciclo de vida simplificado de um processo



7

Processos

- Troca de Contexto / Preempção
  - O processo em execução é suspenso, e um outro processo passa a ser executado
  - Ocorre por determinação do escalonador ou quando o processo que estava sendo executado é suspenso
  - O contexto do processo suspenso deve ser salvo para retomar a execução posteriormente

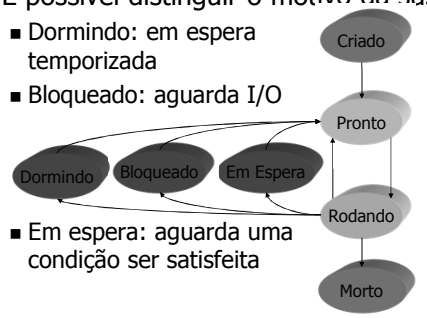
8

Processos

- O contexto de um processo compreende:
  - O estado do processo
  - Informações para escalonamento
  - Dados para contabilização de uso
  - Um segmento de código
  - Um segmento de dados
  - Os valores dos registradores
  - O contador de programa
  - Uma pilha de execução
  - Arquivos, portas e outros recursos alocados

Processos

- É possível distinguir o motivo da suspensão
  - Dormindo: em espera temporizada
  - Bloqueado: aguarda I/O
  - Em espera: aguarda uma condição ser satisfeita



Processos

- Chamadas do Sistema Operacional Unix
  - Criar um Processo:
    - fork() cria uma cópia do processo atual
    - exec() carrega o código do processo

```
...
int pid = fork();            // cria cópia do processo
if  (pid < 0)                // erro na criação
   ...                       // trata o erro
else if (pid > 0)            // é o processo pai
   ...                       // executa código do pai
else if  (pid == 0)          // é o processo filho
   exec("/bin/filho", 0);    // executa código do filho
...
```

11

Processos

- Chamadas do Sistema Operacional Unix
  - Suspender a Execução: sleep(<tempo>) ou wait() → reinicia com kill(<pid>,SIGWAIT)
  - Obter Identificador do Processo: getpid()
  - Aguarda o Fim dos Processos Criados: join()
  - Finalizar o Processo: exit(<retorno>)
  - Destruir um Processo: kill(<pid>,SIGKILL)

12

Processos

- Interação com o Usuário no Unix
  - Processos são criados através da interface gráfica ou de comandos digitados na Shell
  - Comandos bloqueiam a Shell, a não ser que sejam seguidos de um &
  - Os processos em execução são listados com o comando ps –ef (ps –aux em versões antigas)
  - Processos são destruídos com kill -9 <pid>

13

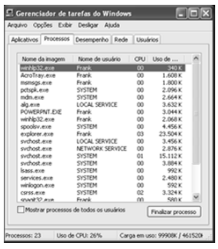
Processos

- Chamadas da API do Windows
  - Criar um Processo: CreateProcess(<nome>, <comando>, ...) ou CreateProcessAsUser(<usuário>,<nome>, ...)
  - Obter o Identificador do Processo: GetCurrentProcessId()
  - Suspender a Execução: Sleep(<tempo>)
  - Finalizar o Processo: ExitProcess(<retorno>)
  - Destruir um Processo: TerminateProcess(<pid>, <retorno>)

14

Processos

- Interação com o Usuário no Windows
  - Processos são criados através da interface gráfica ou de comandos digitados no Prompt
    - O Prompt não bloqueia aguardando aplicações Windows, que executam em outra janela
    - Processos são listados/destruídos pelo Gerenciador de Tarefas



15

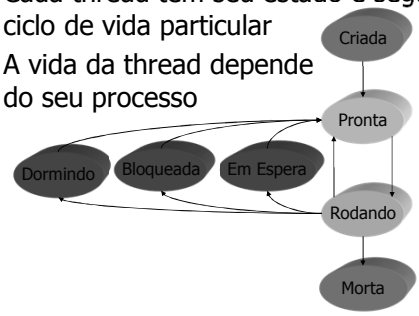
Threads

- Definição
  - *Threads* (linhas) de execução são atividades concorrentes executadas por um processo
  - Um processo pode ter uma ou mais *threads*
  - Threads pertencentes a um mesmo processo compartilham recursos e memória
- Suporte a Threads
  - Threads nativas do S.O.
  - Suporte de programação multi-thread
  - Linguagem de programação multi-threaded

16

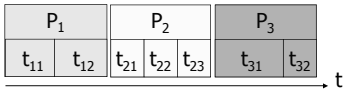
Threads

- Cada thread tem seu estado e segue um ciclo de vida particular
- A vida da thread depende do seu processo

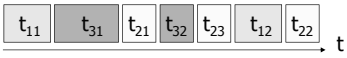


Threads

- Escalonamento
  - Por Processo: escalonador aloca tempo para execução dos processos, que definem como usar este tempo para executar suas threads



  - Por Thread: escalonador define a ordem na qual as threads serão executadas



18

Threads

- Troca de Contexto
  - Quando duas threads de um mesmo processo se alternam no uso do processador, ocorre uma troca de contexto parcial
  - Numa troca parcial, o contador de programa, os registradores e a pilha devem ser salvos
  - Uma troca de contexto parcial é mais rápida que uma troca de contexto entre processos
  - Uma troca de contexto completa é necessária quando uma thread de um processo que não estava em execução assume o processador

Threads

- Threads X Processos

	Processos	Threads
Troca de Contexto	Completa	Parcial
Área de Memória	Independente	Compartilhada
Comunicação	Inter-Processo	Intra-Processo
Código	Independente	Mesmo código
Suporte em S.O.'s	Quase todos	Os mais atuais
Suporte em Ling. Prog.	Quase todas	As mais recentes

20

Threads

- Threads POSIX
  - Padrão adotado pelos UNIX e outros S.O.'s
  - Criar uma Thread: pthread_create(<thread>, <atrib>, <rotina>, <args>);
  - Obter Ident. da Thread: pthread_self()
  - Suspender Execução: pthread_delay_np(<tempo>)
  - Finalizar a Thread: pthread_exit (<retorno>)
  - Linux usa as mesmas rotinas, mas cria um processo por thread → não é 100% POSIX

21

Threads

- Threads no Windows
  - Criar uma Thread: CreateThread (<atribs>, <tam_stack>, <rotina>, <params>, <flags>,<thread_id>)
  - Obter Ident. da Thread: GetCurrentThreadId()
  - Suspender Execução: Sleep(<tempo>) ou SuspendThread (<thread>) → retomar com ResumeThread(<t>) ou SwitchToThread(<t>)
  - Finalizar a Thread: ExitThread(<retorno>)
  - Destruir uma Thread: TerminateThread (<thread>,<retorno>)

22

Paralelismo em Java

- Máquina Virtual Java, Processos e Threads
  - Cada instância da JVM corresponde a um processo do sistema operacional hospedeiro
  - A JVM não possui o conceito de processos – apenas implementa threads internamente
  - Ao ser iniciada, a JVM executa o método main() do programa na thread principal, e novas threads podem ser criadas a partir desta
  - Threads de uma mesma JVM compartilham memória, portas de comunicação, arquivos e outros recursos

Paralelismo em Java

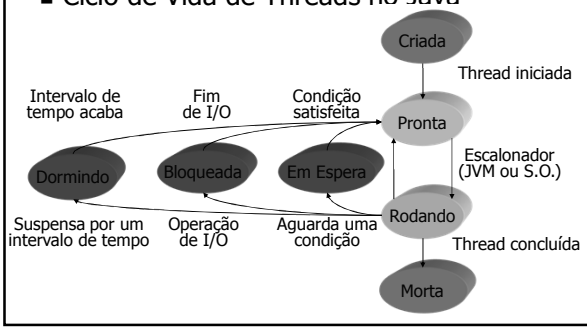
- Implementação de Threads no Java
  - Green Threads
    - Usadas em sistemas sem suporte a *threads*
    - Threads e escalonamento implementados pela máquina virtual Java
  - Threads Nativas
    - Usa threads nativas e escalonador do S.O.
    - Usada nos sistemas suportados oficialmente
      - Linux, Solaris e Windows
    - Ideal em máquinas com >1 processador

Paralelismo em Java

- Estados das Threads em Java
  - Pronta: poderia ser executada, mas o processador está ocupado
  - Rodando: em execução
  - Suspensa:
    - Bloqueada: aguarda operação de I/O
    - Em Espera: aguarda alguma condição
    - Dormindo: suspensa por um tempo

Paralelismo em Java

- Ciclo de Vida de Threads no Java



Paralelismo em Java

- Threads na Linguagem Java
  - O conceito de thread está presente em Java através da classe java.lang.Thread, que é um tipo predefinido da linguagem
  - Java também oferece :
    - Mecanismos para sincronização e controle de concorrência entre threads
    - Classes para gerenciamento de grupos (pools) de threads
    - Classes da API que podem ser acessadas concorrentemente (*thread-safe*)

Paralelismo em Java

- Classe Thread e interface Runnable

```
public class Thread
   extends Object
   implements Runnable {
   ...                    // Métodos da classe thread
}

public interface Runnable {
   public void run();  // Código executado por uma thread
}
```

Paralelismo em Java

- Principais métodos de java.lang.Thread
  - Construtores: Thread(), Thread (Runnable), Thread(String), Thread(Runnable, String), ...
  - Método run(): contém o código executado pela Thread; herdado da interface Runnable; implementado pela Thread ou por um objeto passado como parâmetro para seu construtor
  - Método start(): inicia a Thread em paralelo com a Thread que a criou, executando o código contido no método run()

Paralelismo em Java

- Criando Threads no Java usando herança
  - Definir uma classe que estenda Thread e implemente o método run()

```
public class MyThread extends Thread {
  public void run() {
    ...                                  // código da thread
  }
}
```

  - Criar thread e chamar start(), que executa run()

```
...
MyThread t = new MyThread();       // cria a thread
t.start();                         // inicia a thread
...
```

Paralelismo em Java

- Criando Threads no Java usando herança
  - Definir uma classe que implemente Runnable

```
public class MyRun implements Runnable {
  public void run() {
    ...                                  // código da thread
  }
}
```

  - Criar uma Thread passando uma instância do Runnable como parâmetro e chamar start()

```
...
Thread t = new Thread (new MyRun());  // cria a thread
t.start();                            // inicia thread
...
```

Paralelismo em Java

- Criando Threads no Java sem usar herança
  - Criar um Runnable, definindo o método run(), instanciar a thread passando o Runnable e chamar start()

```
...
Runnable myRun = new Runnable() {  // cria o runnable
   public void run() {
     ...                                 // código da thread
   }
};
...
new Thread(myRun).start();       // cria e inicia a thread
...
```

Paralelismo em Java

- Criando Threads no Java sem usar herança
  - Criar uma Thread, definindo o método run(), e chamar start()

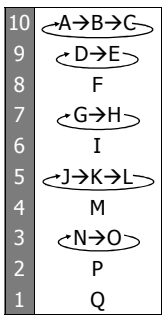
```
...
Thread myThread = new Thread() {  // cria a thread
   public void run() {
     ...                                 // código da thread
   }
};
...
myThread.start();                // executa a thread
...
```

Paralelismo em Java

- Nome da Thread
  - Identificador não-único da Thread
  - Pode ser definido ao criar a Thread com Thread(String) ou Thread(Runnable, String)
  - Se não for definido na criação, o nome da Thread será "Thread-*n*" (com *n* incremental)
  - Pode ser redefinido com setName(String)
  - Pode ser obtido através do método getName()

Paralelismo em Java

- Prioridade da Thread
  - Valor de 1 a 10; 5 é o *default*
  - Thread herda prioridade da thread que a criou
  - Modificada com setPriority(int)
  - Obtida com getPriority()
- Escalonamento
  - Threads com prioridades iguais são escalonadas em *round-robin*, com cada uma ativa durante um *quantum*



Paralelismo em Java

- Outros métodos de java.lang.Thread
  - Obter Referência da Thread em Execução: currentThread()
  - Suspender Execução: sleep(long), interrupt()
  - Aguardar Fim da Thread: join(), join(long)
  - Verificar Estado: isAlive(), isInterrupted()
  - Liberar o Processador: yield()
  - Destruir a Thread: destroy()

Paralelismo em Java

```
public class ThreadSleep extends Thread {
   private long tempo = 0;
   public ThreadSleep(long tempo) { this.tempo = tempo; }  // Construtor
   public void run() { // Código da Thread
      System.out.println(getName() +" vai dormir por "+ tempo +" ms.");
      try {
         sleep(tempo);
         System.out.println(getName() + " acordou.");
      } catch (InterruptedException e) { e.printStackTrace(); }
   }
   public static void main(String args[]) {
      for (int i=0; i<10; i++)
         // Cria e executa as Threads
         new ThreadSleep((long)(Math.random()*10000)).start();
   }
}
```

Paralelismo em Java

- Gerenciando a execução de threads
  - Executores gerenciam grupos de Threads
    - Interfaces Executor e ExecutorService (que estende a primeira; é mais completa)
  - São criados através da classe Executors:
    - SingleThreadExecutor usa uma thread para executar atividades seqüencialmente
    - FixedThreadPool usa um grupo de threads de tamanho fixo para executar atividades
    - CachedThreadPool cria threads sob demanda

Paralelismo em Java

- Exemplo de uso de Executor

```
import java.util.concurrent.*;
public class ExecutorTest {
   public static void main(String args[]) {
      ExecutorService exec = Executors.newSingleThreadExecutor();
//    ExecutorService exec = Executors.newFixedThreadPool(5);
//    ExecutorService exec = Executors.newCachedThreadPool();
      for (int i=0; i<10; i++) {
         // Cria e executa as threads
         exec.execute(new ThreadSleep((long)(Math.random()*10000)));
      }
      // Encerra o executor assim que as threads terminarem
      exec.shutdown();
   }
}
```

Paralelismo em Java

- Escalonamento de atividades
  - Um ScheduledExecutorService permite escalonar a execução de atividades, definindo um atraso para início da atividade e/ou um período de repetição entre execuções
    - SingleThreadScheduledExecutor usa uma thread para todas as atividades escalonadas
    - ScheduledThreadPool cria um grupo de threads para executar as atividades

Paralelismo em Java

- Escalonamento com ScheduledExecutor

```
import java.util.concurrent.*;
public class ScheduleThread extends Thread {
   public void run () { System.out.println(getName() + " executada.");
   }
   public static void main(String args[]) {
      ScheduledExecutorService exec =
                  Executors.newSingleThreadScheduledExecutor();
      for (int i=0; i<10; i++) {
         long tempo = (long) (Math.random()*10000);
         System.out.println("Thread-" + i
                           + " será executada em "+ tempo + "ms.");
         // Escalona a execução da thread
         exec.schedule(new ScheduleThread(),
                        tempo, TimeUnit.MILLISECONDS);
      }
      exec.shutdown();
   }}
```

Paralelismo em Java

- Execução periódica com ScheduledExecutor

```
import java.util.concurrent.*;
public class CountdownThread extends Thread {
   private static long tempo = 0, cont = 10, espera = 5, intervalo = 1;
   public CountdownThread(long tempo) { this.tempo = tempo; }
   public void run () {
      System.out.println(tempo +"segundos para o encerramento.");
      tempo--; }
   public static void main(String args[]) {
      ScheduledExecutorService exec =
                  Executors.newSingleThreadScheduledExecutor();
      exec.scheduleAtFixedRate(new CountdownThread(cont),
                              espera, intervalo, TimeUnit.SECONDS);
      try { exec.awaitTermination(cont+espera, TimeUnit.SECONDS);
      } catch (InterruptedException ie) { ie.printStackTrace(); }
      exec.shutdown();
   }
}
```

Paralelismo em Java

- Grupos de Threads
  - A classe ThreadGroup define um grupo de Threads que são controladas conjuntamente
  - O grupo da Thread é definido usando o construtor Thread(ThreadGroup, ...)

```
// Cria grupo de Threads
ThreadGroup myGroup = new ThreadGroup("MyThreads");
...
// Cria Threads e as insere no grupo
Thread myThread1 = new MyThread(myGroup,"MyThread"+i);
Thread myThread2 = new MyThread(myGroup,"MyThread"+i);
...
// Interrompe todas as Threads do grupo
group.interrupt();
...
```

Paralelismo em Java

- Variáveis locais em Threads
  - A classe ThreadLocal permite criar variáveis locais para Threads (ou seja, que têm valores distintos para cada Thread)

```
// Cria variável local
static ThreadLocal valorLocal = new ThreadLocal();
...
// Define o valor da variável para esta Thread
valorLocal.set(new Integer(10));
// Obtém o valor de var correspondente a esta Thread
int valor = ((Integer) valorLocal.get()).intValue();  // valor = 10
// Outra thread pode acessar a mesma variável e obter outro valor
int valor = ((Integer) valorLocal.get()).intValue();  // valor = 0
```