Processamento Paralelo com Promises

MAC 5742 - Computação Paralela e Distribuída
Prof. Alfredo Goldman

Mateus Espadoto

10 de junho de 2015

Promises - Introdução

- Abstração para programação concorrente:
  - Chamadas assíncronas sem uso explícito de threads
  - Composição de tarefas
  - Controle de exceções
- Sintaticamente mais limpo do que programar com threads ou callbacks
- Proposto originalmente por Friedman e Wise em 1976
  - *The Impact of Applicative Programming on Multiprocessing. International Conference on Parallel Processing.*

Promises - Terminologia

- **Promise**: uma referência para um valor que ainda não é conhecido e que permite:
  - Monitorar estado da execução
  - Obter valor final
  - Encadeamento de chamadas para eventos de término com sucesso ou falha
- **Deferred**: uma computação não concluída, que cumprirá a promessa com um valor ou uma exceção
- Exemplo:

```
var promise = async deferred(args)
```

Promises - Terminologia

- Não há padronização de nomes entre linguagens
  - Javascript: Promise e Deferred
  - Java: FutureTask (Runnable) e Callable
  - Scala: Future e Promise
  - Python: Future e callable (qualquer função/método)
  - Microsoft: Task e Action
- Confusão similar ao mundo pré-pthreads no Unix (threads vs lightweight processes)

Promises - Estados possíveis

- Promises podem ter os seguintes estados:
  - **Pendente** (em execução)
  - **Resolvida** (concluída com sucesso)
  - **Rejeitada** (concluída com falha)
- Promises resolvidas devem conter um valor final
- Promises rejeitadas devem conter o motivo da falha
- Falhas (rejeições) podem ser tratadas localmente ou globalmente
- Em uma cadeia, o valor final da promise precedente é passado para a promise subsequente

Promises - Exemplos

- Exemplos em Javascript com a biblioteca Q.js
- **Chamada assíncrona**

```
var promise = Q.fcall(A);
```

- **then(onFulfilled, onRejected)**
  - Encadeamento de chamadas
  - Chamada ao próximo método da cadeia
  - Tratamento de erro local, lógica de retry

```
Q.fcall(A).then(B, falhaA).then(C, falhaB);
```

Promises - Exemplos

- **catch(errorHandler)**
  - Tratamento de erro global
  - Similar a um bloco try/catch assíncrono

```
Q.fcall(A).then(B).then(C).then(D).catch(errorHandler);
```

- **all([f1, f2, . . . , fn])**
  - Cria uma promise que é cumprida quando todas as suas sub-promises forem cumpridas

```
Q.all([A, B, C]).done(D);
```

- **any([f1, f2, . . . , fn])**
  - Cria uma promise que é cumprida pela primeira das sub-promises que for cumprida

```
Q.any([A, B, C]).done(D);
```

Promises - Casos de Uso

- Funcionalidades normalmente implementadas com threads
  - Chamadas assíncronas
  - Barreiras
- Composição de linhas de execução concorrentes
  - Encadeamento de métodos/funções existentes
  - Tratamento de exceção externo ao método
- Divisão de trabalho (ex: laço for paralelo)
  - Similar ao uso de threads, sem grandes vantagens
  - Linguagens possuem outros recursos para isso, similar a OpenMP
- Javascript: Redução do "Callback Hell"

Promises - Disponibilidade

- Disponibilidade em linguagens (alguns exemplos):
  - Javascript (Q.js, when.js, jQuery)
  - Java (java.util.concurrent.Future, JDeferred)
  - C# (System.Threading.Tasks.Task)
  - Python (concurrent.futures, aplus)
  - Scala (scala.concurrent)
- Iniciativa de padronização: Promises/A+
  - Javascript
  - Existem implementações A+ para outras linguagens
    - Java: JDeferred
    - Python: aplus

Promises - Referências

- Parallel Processing with Promises, Communications of the ACM Maio/2015, pag. 42-47
- Wikipedia
- Promises/A+
- JDeferred
- aplus
- Documentação das Linguagens
  - Java
  - Scala
  - Python
  - Microsoft