



Embarcadero Conference

embarcadero®

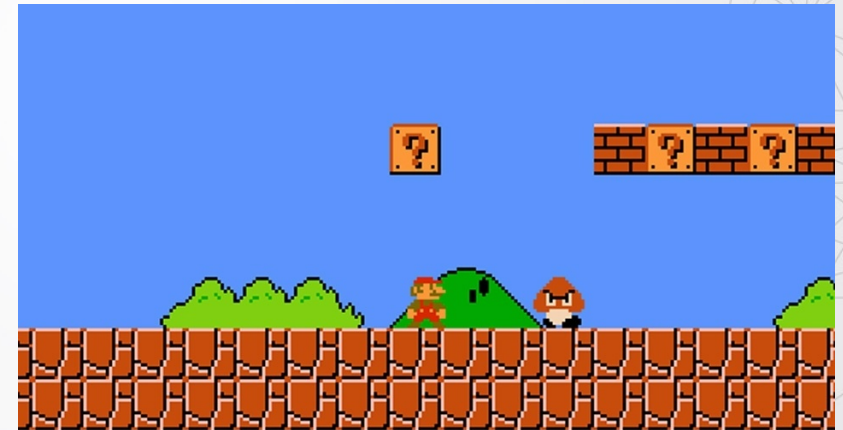


Delphi Parallel Programming Library

Mário Guedes – jmarioguedes@gmail.com

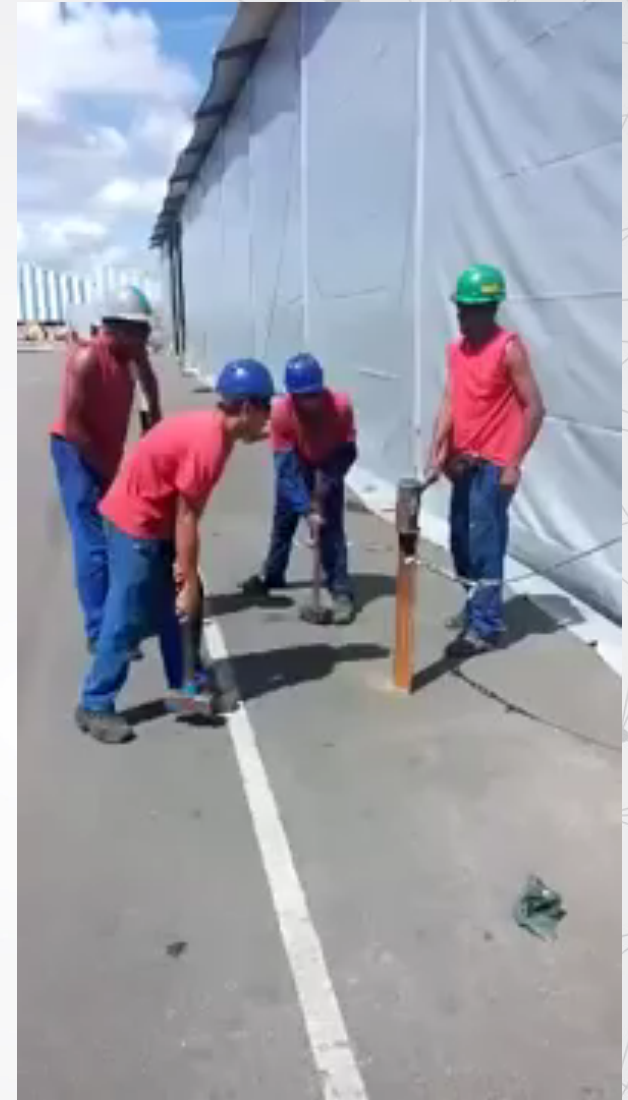
Mário?

- Gerente de Desenvolvimento na Contact Studio Software
 - *Estamos contratando!*
 - <http://www.g4solutions.com.br/trabalhe-conosco/>
- Desenvolvedor Delphi, Python, JS e noSQL
- 15+ anos na lida
- Filho de Valdete e Joselito
- Irmão do Manoel, da Jenny e do Jonhy
- Pai do Júlio e da Fernanda
- E noivo da Tamires



PPL

- Biblioteca de Programação Paralela
- Presente desde o Delphi XE7
- Recurso nativo da linguagem (RTL)
- Multiplataforma!
 - Seja no mobile, seja no Win32 – o código é o mesmo.
- *Sexy sem ser vulgar: SMART*



Principal ganho

- Deixamos de criar threads explicitamente
- Sem perder a chance de paralelizar as tarefas
- Isso diminui a dispersão de regras nos códigos
- E ganhamos aplicativos mais responsivos!
- *Comece a pensar em tarefas, tarefas paralelas.*
 - *Petar Georgiev*

Síncrono X Assíncrono

- Vamos fazer valer esse monte de processador aí na sua máquina ou dispositivo?
 - Cálculo complexo
 - Downloads
 - Processamento de arquivos
 - Consumo de serviços REST
 - Consulta à banco de dados
- *Tudo isto de forma não bloqueante!*

Arsenal

- Unit mágica: **System.Threading**
- Três usos primários:
 - ThreadPool
 - Task e ITask
 - IFuture
 - Paralelismo em estrutura de repetição

TThreadPool.Default

- Conjunto de threads prontas a lhe servir!
- Possui um mecanismo sofisticado de gestão destas threads.
 - Algoritmo “Work Stealing Thread Queue”
- `.QueueWorkItem()` – Enfileira uma tarefa para execução.
Pode ser passado:
 - `TNotifyEvent`
 - `TProc` – Método anônimo
- Podemos consultar ou definir a quantidade de trabalhadores com:
 - `.MinWorkerThreads()` - `SetMinWorkerThreads()`
 - `.MaxWorkerThreads()` - `SetMaxWorkerThreads()`

TParallel

- Tem por objetivo paralelizar tarefas a uma só vez. É uma classe selada portanto não é o caso de descender dela.
- `.Join()` – Executa um conjunto de tarefas, devolvendo um `ITask`. Utiliza o `.For()` por isso o grau de paralelismo é determinado internamente.
- `.For()` – Executa um laço `for...to...do`, sob um range executando uma tarefa para cada iteração.

TTask, ITask, IFuture

- TTask é uma classe conveniente para tirar proveito das interfaces ITask e IFuture;
- .Create() – Retorna uma tarefa não agendada
- .Run() – Agenda um ITask
- .Future<T> - Agenda um IFuture
- .CurrentTask – Retorna a tarefa corrente, conveniente para verificar o estado dela
- .WaitForAll – Aguarda o término de todas as tarefas dentro de um array
- .WaitForAny – Aguarda o término de ao menos uma tarefa dentro de um array

Dicas e Cuidados

- Conhecimentos mais profundos em Threads nos trará mais possibilidades. Saiba mais em:
 - http://eugostododelphi.blogspot.com.br/2016/10/material-sobre-threads_51.html
- Cuidado com o escopo das variáveis. Se mostrou mais seguro criar uma função que retorne um método anônimo.
- Certamente a biblioteca evoluirá com o passar do tempo, vamos evoluir juntos?


@ jmarioguedes@gmail.com

@ mario.guedes@contactstudio.com

 /jmarioguedes

 /jmarioguedes

[OBRIGADO]

Embarcadero  onference