UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

# BCC35A - Linguagens de Programação

Prof. Dr. Rodrigo Hübner

## Aula 12: Concorrência

# Introdução

- Concorrência pode acontecer em quatro níveis:
  - Instrução de máquina
  - Instrução de linguagem
  - Unidade (subprograma)
  - Programa completo

# Introdução

- **Linha de controle** (*thread of control*): pontos de programa no qual um controle está fluindo
- Várias linhas de controle em um programa: ***multithreaded***
- Motivações:
  - Aumentar a **velocidade de execução**
  - Muitas situações do mundo real envolvem concorrência
  - Ser **escalável**
  - **Computadores com múltiplos processadores** são amplamente usados

# Concorrência em subprogramas

- **Tarefa** ou **processo** é uma unidade que pode ser concorrente com outras unidades do programa
- Sincronização de tarefas
  - Sincronização de **cooperação**
  - Sincronização de **competição**
- Comunicação é necessária para a sincronização
  - Variáveis não locais compartilhadas
  - Parâmetros
  - Troca de mensagem

# Exemplos de concorrência em LPs

- `Thread Pool` em `C++`
- `Dask` em `Python` para tarefas *Multicore*
- `Worker Threads` em `Javascript`

# `Thread Pool` em `C++`

- Cria uma lista de trabalhadores **fixa** ao invés de gerenciar diversas *threads* em execução.

```
ThreadPool pool(8);
...
pool.enqueue(...);
```

# `Dask` em `Python` *Multithreaded*

- Dados de bibliotecas mais utilizadas são divididos em *chunks*
  - `numpy`: `array`
  - `pandas`: `dataframe`

```
import dask.array as da

img = da.ones((100, 1000, 1000), chunks=(10, 100, 100))
x = da.random.random((10000, 10000))
```

# `Worker Threads` em `Javascript`

- Utiliza o conceito de "Cliente-Servidor" para a submissão e comunicação de tarefas assíncronas.
- `main.js`:

```
const { Worker } = require('worker_threads');

const worker = new Worker('./worker.js');       // cria o worker
worker.on('message', message => {               // recebe msg
  console.log('Mensagem recebida:', message);
})
worker.postMessage('Olá worker!')               // envia msg
```

# Worker Threads em Javascript

- `worker.js`:

```
const { parentPort } = require('worker_threads');

// Recebendo mensagens da thread principal
parentPort.on('message', message => {
  console.log('Mensagem recebida da thread principal:', message)
  parentPort.postMessage('Olá da thread worker!')
});
```

```
>>> end()
```

```
That's all Folks!
```