INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE MATO GROSSO DO SUL
CÂMPUS AQUIDAUANA
CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS PARA INTERNET

MATHEUS DANIEL CRISTAL COMPAROTTO GOMES
NATHAN ALVES GAUNA
PAULO DANIEL MORAES RIBEIRO

GERENCIAMENTO DE PROCESSOS E THREADS EM TYPESCRIPT: UMA ANÁLISE PRÁTICA

AQUIDAUANA - MS
2023

MATHEUS DANIEL CRISTAL COMPAROTTO GOMES

Título: Gerenciamento de processos e threads em TypeScript: uma análise prática.

Relatório apresentado no Curso Superior de Tecnologia em Sistemas para Internet do Instituto Federal de Educação, Ciência e Tecnologia de Mato Grosso do Sul como requisito para obtenção da nota parcial das atividades da unidade curricular Sistemas Operacionais I.

Orientador: Prof. Me. Genair Christo Viana.

AQUIDAUANA - MS

2023

RESUMO

Este trabalho tem como objetivo analisar práticas de gerenciamento de processos e threads em aplicações desenvolvidas em TypeScript. Para isso, serão apresentados conceitos teóricos de gerenciamento de processos e threads, bem como módulos disponíveis em TypeScript para implementação de soluções. Serão apresentados também exemplos práticos de implementação de processos e threads em TypeScript, com a finalidade de demonstrar as possibilidades e limitações dessa abordagem.

Palavras-chave: Processos; Threads; TypeScript; Assincronismo; Benchmarking.

LISTA DE CÓDIGOS

SUMÁRIO

1. INTRODUÇÃO

O gerenciamento de processos e threads é um tema fundamental para a eficiência e desempenho dos sistemas computacionais. Eles são responsáveis por gerir as atividades e tarefas executadas por um programa, permitindo que múltiplas operações sejam realizadas simultaneamente.

Além disso, a linguagem TypeScript tem ganhado cada vez mais popularidade entre os desenvolvedores de software devido à sua tipagem estática e outras características que visam aumentar a qualidade e produtividade do código. Neste trabalho, iremos analisar a utilização do TypeScript para gerenciamento de processos e threads, explorando sua aplicação prática e vantagens.

2. OBJETIVOS

Este trabalho tem como objetivo analisar o gerenciamento de processos e threads utilizando a linguagem de programação TypeScript. A seguir, serão apresentados os objetivos gerais e específicos que nortearão o desenvolvimento do trabalho.

2.1. OBJETIVOS GERAIS

- Analisar a utilização da linguagem TypeScript para gerenciamento de processos e threads;
- Discutir os benefícios e desafios do uso de TypeScript para essa finalidade;
- Apresentar exemplos de aplicações práticas que utilizam TypeScript para gerenciamento de processos e threads.

2.2. OBJETIVOS ESPECÍFICOS

- Explorar as funcionalidades de gerenciamento de processos e threads em TypeScript;
- Investigar o uso do módulo `child_process` do Node.js em TypeScript;
- Implementar um exemplo prático de manipulação de processos em TypeScript;
- Avaliar a qualidade do código produzido em TypeScript para gerenciamento de processos e threads.

3. METODOLOGIA

A seção de metodologia deste trabalho tem como objetivo descrever os procedimentos e ferramentas utilizados na análise do gerenciamento de processos e threads em uma aplicação desenvolvida em TypeScript. O levantamento bibliográfico foi utilizado para coletar informações teóricas sobre o gerenciamento de processos e threads, bem como sobre as funcionalidades oferecidas por módulos como `child_process` do Node.js e outras bibliotecas relevantes para implementação de soluções.

Foi criado um repositório intitulado "sort-methods-runtime-register" na plataforma de hospedagem de código-fonte e arquivos com controle de versão GitHub, disponível em <https://github.com/mdccg/sort-methods-runtime-register>, o qual consiste em uma aplicação para salvar o tempo de execução do método de ordenação Quick Sort em um arquivo de texto plano para um vetor com dez mil elementos aleatórios. Cada elemento é gerado pela função `generateNumber`, exemplificada abaixo:

Código 1 - Função de gerar um número aleatório.

```
1. const generateNumber = (
2.   min: number = Number.MIN_SAFE_INTEGER,
3.   max: number = Number.MAX_SAFE_INTEGER
4. ) => {
5.   return min + Math.floor(Math.random() * (max - min));
6. }
```

Fonte: acervo pessoal.

O retorno desta função é um número inteiro aleatório de 32 bits. Em notação matemática, um número x que obedece às restrições $-2^{31} \leq x \leq 2^{31} - 1$ e $x \in Z$. Tal função foi utilizada somente na função `generateNumberArray`, a qual gera um vetor numérico de tamanho definido por parâmetro.

Código 2 - Função de gerar um vetor numérico com dez mil elementos.

```
1. export const generateNumberArray = (
2.   length: number = 1e4
3. ): number[] => {
4.   return Array(
5.     length
6.   ).fill(0).map(() => generateNumber());
```

```
7. }
```

Fonte: acervo pessoal.

A função foi exportada para ser utilizada no código principal. Para testar a função, foi criada outra função para salvar o vetor numérico em um arquivo de texto plano intitulado "array.txt". A função utiliza apenas os módulos nativos do Node.js `path` e `fs` para criar ou localizar o arquivo em questão e alterar seu conteúdo.

Código 3 - Função de salvar o vetor numérico de dez mil elementos em um arquivo de texto plano.

```
1. import { join } from 'path';
2. import { writeFileSync } from 'fs';
3.
4. export const saveArray = (numberArray: number[]): void
=> {
5.       const path = join(__dirname, '..', 'data',
'array.txt');
6.   const data = numberArray.join('\n');
7.   writeFileSync(path, data);
8.   console.log(`Vetor gravado no arquivo "array.txt".`);
9. }
```

Fonte: acervo pessoal.

No código principal, as funções `generateNumberArray` e `saveArray` são importadas. Então, o vetor numérico é gerado e armazenado na constante[1] `numberArray`. Por fim, são utilizados os métodos `console.time` e `console.timeEnd`.

Código 4 - Código principal.

```
1.     import     {     generateNumberArray     }     from
'./src/functions/generateArray';
2. import { saveArray } from './src/functions/saveArray';
3.
4. const numberArray = generateNumberArray();
5.
6. saveArray(numberArray);
7.
8. console.time('Quick sort');
```

[1] Devido a uma particularidade das linguagens de programação JavaScript e TypeScript, objetos como vetores, matrizes, estruturas de dados *etc*. são mutáveis mesmo que sejam declarados como constantes.

```
9. numberArray.sort();
10. console.timeEnd('Quick sort');
```

Fonte: acervo pessoal.

O método `console.timeEnd` imprime a string “Quick sort:” concatenada ao tempo de execução do retalho de código — no caso do código principal, apenas a linha 9 — em milissegundos. Entretanto, o mesmo método não tem retorno, o que faz com que não seja possível capturar o tempo de execução do retalho de código em uma variável ou constante. A solução é escrever um código “gravador” que será executado como um processo pelo sistema operacional, executará o código principal como um processo filho e o processo filho será vinculado a uma thread em seu fluxo de saída padrão de dados que capturará a saída do processo filho.

Código 5 - Código gravador.

```
1. import { spawn } from 'child_process';
2. import { join } from 'path';
3. import { writeFileSync, appendFileSync } from 'fs';
4.
5. const command = 'yarn';
6. const options = ['start'];
7.
8. const childProcess = spawn(command, options);
9.
10.   const   path   =   join(__dirname,   '..',   'data',
'output.txt');
11. writeFileSync(path, '');
12.
13. childProcess.stdout.on('data', (data) => {
14.   appendFileSync(path, `${data}`);
15. });
16.
17. childProcess.on('exit', (code: number | null, signal:
NodeJS.Signals | null) => {
18.    console.log(`O processo filho encerrou com código
${code} e sinal ${signal}`);
19. });
```

Fonte: acervo pessoal.

Por fim, no arquivo “output.txt”, poderá ser encontrado o seguinte conteúdo:

Código 6 - Saída do código principal gravada por meio de uma thread.

```
1. $ ts-node $npm_package_main
2. Vetor gravado no arquivo "array.txt".
3. Quick sort: 16.622ms
```

Fonte: acervo pessoal.

4. CONSIDERAÇÕES FINAIS

Neste trabalho, foi possível realizar uma análise das práticas de gerenciamento de processos e threads em aplicações desenvolvidas em TypeScript. A partir da revisão bibliográfica, foram apresentados conceitos teóricos relevantes para o entendimento do tema, bem como módulos disponíveis em TypeScript para a implementação de soluções.

Os exemplos práticos apresentados mostraram as possibilidades e limitações dessa abordagem, demonstrando como é possível melhorar o desempenho e a eficiência de uma aplicação utilizando processos e threads.

Por fim, espera-se que este trabalho possa contribuir para a compreensão e utilização de técnicas de gerenciamento de processos e threads em aplicações desenvolvidas em TypeScript, incentivando novas pesquisas e práticas que possam aprimorar ainda mais essa área de conhecimento.

5. REFERÊNCIAS BIBLIOGRÁFICAS

- BRIDGE, Karl; SHARKEY, Kent; SATRAN, Michael. **Processes and threads**. [*S. l.*]: Microsoft, 24 set. 2022. Disponível em: https://learn.microsoft.com/pt-br/windows/win32/procthread/processes-and-threads. Acesso em: 7 mar. 2023;
- MANIERO, Antonio. **Terminologia**: Existe diferença entre Programa, Thread e Processo?. [*S. l.*]: StackOverflow, 29 mai. 2016. Disponível em: https://pt.stackoverflow.com/a/131122. Acesso em: 7 mar. 2023;
- BRIDGE, Karl et al. **Processor groups**. [*S. l.*]: Microsoft, 21 set. 2022. Disponível em: https://learn.microsoft.com/pt-br/windows/win32/procthread/processor-groups. Acesso em: 7 mar. 2023;
- BRIDGE, Karl et al. **Multiple processors**. [*S. l.*]: Microsoft, 21 set. 2022. Disponível em: https://learn.microsoft.com/pt-br/windows/win32/procthread/multiple-processors. Acesso em: 8 mar. 2023;
- BRIDGE, Karl et al. **Thread pools**. [*S. l.*]: Microsoft, 24 set. 2022. Disponível em: https://learn.microsoft.com/pt-br/windows/win32/procthread/thread-pools. Acesso em: 8 mar. 2023;
- BRIDGE, Karl et al. **What's new in processes and threads**. [*S. l.*]: Microsoft, 21 set. 2022. Disponível em: https://learn.microsoft.com/pt-br/windows/win32/procthread/what-s-new-in-processes-and-threads. Acesso em: 8 mar. 2023.