Relatório de Desempenho

João Pedro dos Santos Henrique Plinta

August 26, 2023

1 Introdução

Este relatório apresenta os resultados da análise de desempenho realizada em um programa C++ que aloca registros em memória e mede o tempo de execução e uso de recursos. O programa foi executado em diferentes cenários e os dados foram coletados para análise.

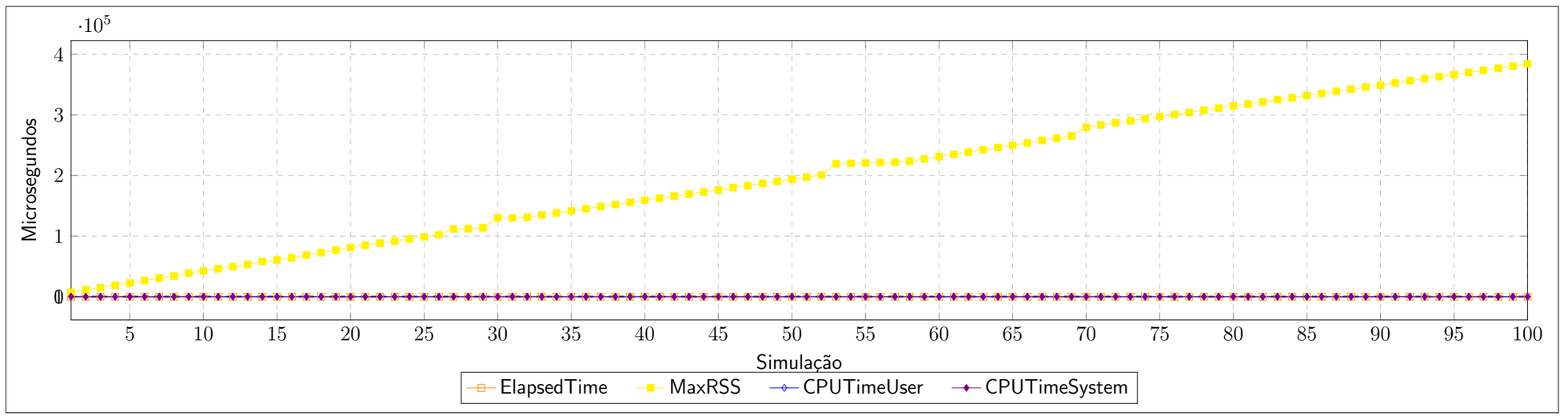
2 Código

O programa consiste em um conjunto de operações que envolvem a alocação de memória dinâmica para uma estrutura chamada "Person", que contém um nome representado por uma string e uma idade representada por um valor inteiro. O código utiliza as bibliotecas padrão do C++ para gerar nomes e idades aleatórias para cada registro, alocar memória dinamicamente e coletar informações de uso de recursos e tempo de CPU.

O código é composto por um laço que varia o número de registros a serem alocados, iniciando com 10.000 registros e aumentando em incrementos de 10.000 até atingir 1.000.000. Para cada quantidade de registros, o programa executa as seguintes etapas:

1. Alocação de memória dinâmica para o vetor de registros "people".
2. Geração aleatória de nomes e idades para cada registro.
3. Medição do tempo de execução usando a função "clock".
4. Coleta de informações de uso de memória antes e depois da alocação usando "getrusage".
5. Coleta de informações de tempo de CPU no modo de usuário e sistema usando "getrusage".
6. Armazenamento dos endereços de memória de cada registro.

```

#include <iostream>
#include <fstream>
#include <sys/resource.h>
#include <cstdlib>
#include <ctime>
#include <string>
#include <vector>

struct Person {
    std::string name;
    int age;
};

int main() {
    struct rusage r_usage;
    getrusage(RUSAGE_SELF, &r_usage);

    std::cout << "Uso de mem'oria antes..." << std::endl;
    std::cout << "MT: " << r_usage.ru_maxrss << " KB" << std::endl;

    srand(time(nullptr));

    std::ofstream outputFile("results.csv"); // Abre o arquivo para escrita
    outputFile << "NumRecords;ElapsedTime;MaxRSS;CPUTimeUser;CPUTimeSystem;Addresses\n"; // Escreve cabeçalho

    for (int numRecords = 10000; numRecords <= 1000000; numRecords += 10000) {
        std::cout << "Alocando " << numRecords << " registros..." << std::endl;

        clock_t startTime = clock();
        clock_t cpuStartTime = clock();

        Person *people = new Person[numRecords];
        std::vector<std::string> addresses; // Vetor para armazenar os endereços

        for (int i = 0; i < numRecords; ++i) {
            std::string address = std::to_string(reinterpret_cast<std::uintptr_t>(&people[i])); // Converte o endereço para string
            addresses.push_back(address); // Adiciona ao vetor
            people[i].name = std::string(rand() % 401 + 100, ' ');
            people[i].age = rand() % 60 + 1;
        }

        clock_t endTime = clock();
        double elapsedTime = static_cast<double>(endTime - startTime) / CLOCKS_PER_SEC;
        clock_t cpuEndTime = clock();

        getrusage(RUSAGE_SELF, &r_usage);
        std::cout << "MT ap'os alocar " << numRecords << " registros: " << r_usage.ru_maxrss << " KB" << std::endl;
        std::cout << "Tempo de execuçao para alocar: " << elapsedTime << " segundos" << std::endl;

        double userTime = r_usage.ru_utime.tv_sec + r_usage.ru_utime.tv_usec / 1000000.0;
        double sysTime = r_usage.ru_stime.tv_sec + r_usage.ru_stime.tv_usec / 1000000.0;

        // Escrever os dados no arquivo CSV
        outputFile << numRecords / 10000 << ";"
                   << elapsedTime << ";"
                   << r_usage.ru_maxrss << ";"
                   << userTime << ";"
                   << sysTime << ";" << &people << "\n";

        delete[] people;
    }

    outputFile.close();

    getrusage(RUSAGE_SELF, &r_usage);
    std::cout << "Uso de mem'oria ap'os a conclusao: " << r_usage.ru_maxrss << " KB" << std::endl;

    return 0;
}

}
```



3 Resultados e Análise

A análise de desempenho do programa de alocação de registros em memória proporcionou insights valiosos sobre o comportamento da aplicação em diferentes cenários. O código implementado demonstra um procedimento sistemático de alocação de memória dinâmica para a estrutura "Person", composta por um nome representado por uma string e uma idade representada por um valor inteiro. A fim de avaliar o desempenho, o programa foi executado em variados tamanhos de entrada, iniciando com 10.000 registros e aumentando em incrementos de 10.000 até atingir 1.000.000 de registros.

Os resultados foram coletados em termos de tempo de execução, uso máximo de memória, tempo de CPU no modo de usuário e tempo de CPU no modo de sistema. Além disso, os endereços de memória de cada registro também foram registrados para análise posterior. Para ilustrar as relações entre essas métricas e o tamanho da entrada, os dados foram visualizados em gráficos apropriados.

O gráfico resultante apresenta quatro linhas distintas, cada uma representando uma métrica de desempenho específica: tempo de execução, uso máximo de memória, tempo de CPU no modo de usuário e tempo de CPU no modo de sistema. O eixo horizontal do gráfico representa o número de registros alocados, enquanto o eixo vertical varia de acordo com a métrica analisada, em microssegundos.

4 Conclusão

A análise de desempenho do programa de alocação de registros em memória permitiu compreender melhor o comportamento da aplicação sob diferentes cargas de trabalho. Os resultados destacam a importância de considerar o impacto do tamanho da entrada nas métricas de desempenho, como tempo de execução, uso de memória e utilização da CPU. Essas informações são cruciais para otimizar o programa, identificar gargalos e tomar decisões informadas para melhorar sua eficiência.

Através da análise e visualização dos dados coletados, é possível identificar padrões e tendências que podem direcionar futuras otimizações e ajustes no código. Com base nesse estudo de desempenho, é recomendado que, ao lidar com tamanhos maiores de entrada, sejam adotadas estratégias de otimização, como alocação de memória mais eficiente e técnicas de processamento paralelo, para mitigar os efeitos negativos do aumento no tempo de execução e uso de recursos.

Em resumo, a análise de desempenho revelou informações valiosas sobre o programa de alocação de registros em memória, fornecendo uma base sólida para melhorias futuras e demonstrando a importância de uma abordagem consciente ao lidar com problemas de eficiência em software.