Relatório "prática de alocação de memória"

Gabriel de Paula Gaspar Pinto

07 outubro 2024

1 Introdução

Este relatório contém os resultados de um programa em C++, que foi desenvolvido com o intuito de aprendizagem em relação à alocação dinâmica de memória, da matéria de Estrutura de Dados, utilizando grandes quantidades de dados. Assim, é possível entender como a alocação de memória com grandes quantidades afetam o sistema.

2 Ambiente de desenvolvimento

Este programa foi desenvolvido e testado em um computador "Samsung Galaxy Book 4 Pro", utilizando o Windows Subsystem for Linux (WSL), na edição Linux 5.15.153.1-microsoft-standard-WSL2, com as seguintes especificações:

- Sistema Operacional: Windows 11
- Processador: Intel Core Ultra 7 155H
- Memória: 16GB 7467MT/s
- IDE: Visual Studio Code 1.94.0

3 Código C++

```cpp
#include <iostream>
#include <iomanip>
#include <string>
#include <random>
#include <sys/resource.h>
using namespace std;

//Estrutura generica requerida no enunciado
struct Generico {
    string text;
    double numA;
    double numB;
} ;

//coisas/iniciador para a biblioteca random funcionar
const string CHARS = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
random_device rd;
mt19937 gen(rd());
uniform_int_distribution <> texto(0, CHARS.size() -1);
uniform_real_distribution <> valorA(1, 10000000);           //possui dois geradores de numeros reais
uniform_real_distribution <> valorB(1, 10000000);           //para nao gerarem o mesmo numero
uniform_int_distribution <> qtd(100, 500);

//funcao para gerar uma string aleatoria, entre 100 a 500 caracteres
string randomStringGen (int length) {
    string randomString;
    for (long int j = 0; j < length; ++j) {
        randomString += CHARS[texto(gen)];
    }
    return randomString;
}

//funcao para medir o tempo do programa
//copiado do programa_mede_memoria.cpp e transformado em funcao
```

```
35 void tempoDeUso() {
36     struct rusage usage;
37     getrusage(RUSAGE_SELF, &usage);
38     cout << endl;
39     cout << "Tempo de usuario (user): " << usage.ru_utime.tv_sec + usage.ru_utime.tv_usec / 1000000.0
       << "s" << endl;
40     cout << "Tempo de sistema (sys): " << usage.ru_stime.tv_sec + usage.ru_stime.tv_usec / 1000000.0 <<
        "s" << endl;
41     cout << endl;
42 }
43
44 int main() {
45
46     long int i = 0;
47
48     struct rusage medeMemoria;
49     getrusage(RUSAGE_SELF, &medeMemoria);
50     cout << "Uso de memoria antes do programa: " << medeMemoria.ru_maxrss << " KB" << endl;
51
52     Generico* generico = new Generico;
53
54     tempoDeUso();
55
56     //loop que gera 10000 strings (entre 100 a 500 caracteres) e 2 valores reais (doubles)
57     for (i = 0; i < 10000; i++) {
58
59         Generico* generico = new Generico;
60         generico->text = randomStringGen(qtd(gen));
61         generico->numA = valorA(gen);
62         generico->numB = valorB(gen);
63
64     }
65
66     tempoDeUso();
67
68     //loop que vai ate 1e6
69     for (i = 10000; i <= 1000000; i += 5000) {
70
71         for (int j = 1; j <= 5000; j++) {
72
73         Generico* generico = new Generico;
74         generico->text = randomStringGen(qtd(gen));
75         generico->numA = valorA(gen);
76         generico->numB = valorB(gen);
77
78         }
79
80         getrusage(RUSAGE_SELF, &medeMemoria);
81         cout << "Processo: " << i << endl;
82         cout << "Uso de memoria atual: " << medeMemoria.ru_maxrss << " KB" << endl;
83         cout << "Endereco: " << &generico << "\n \n";
84
85     }
86
87     tempoDeUso();
88
89     getrusage(RUSAGE_SELF, &medeMemoria);
90     cout << "Uso de memoria depois do programa: " << medeMemoria.ru_maxrss << " KB" << endl;
91
92     delete generico;
93
94     cout << "\n" << "confirmacao que o codigo terminou de rodar, depois de deletar" << endl;
95
96     return 0;
97 }
```

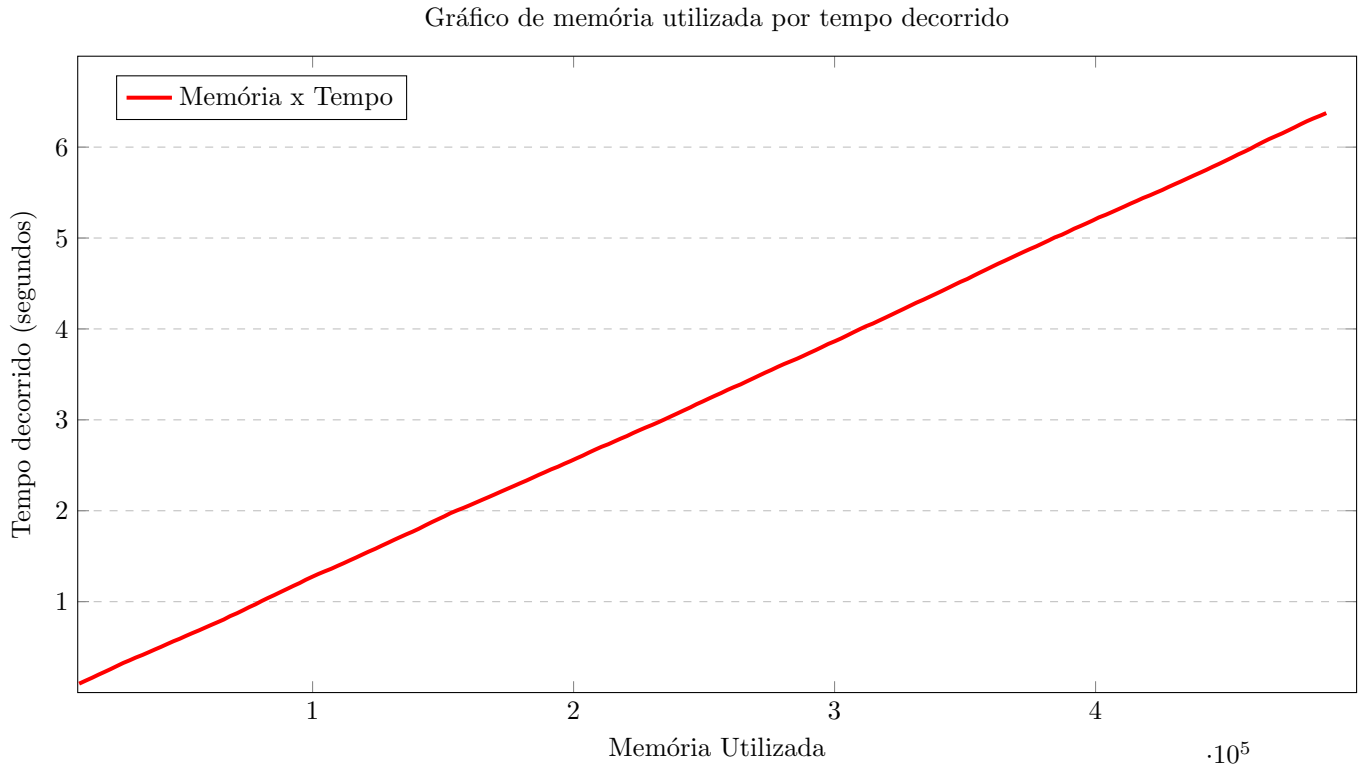
4 Desafios

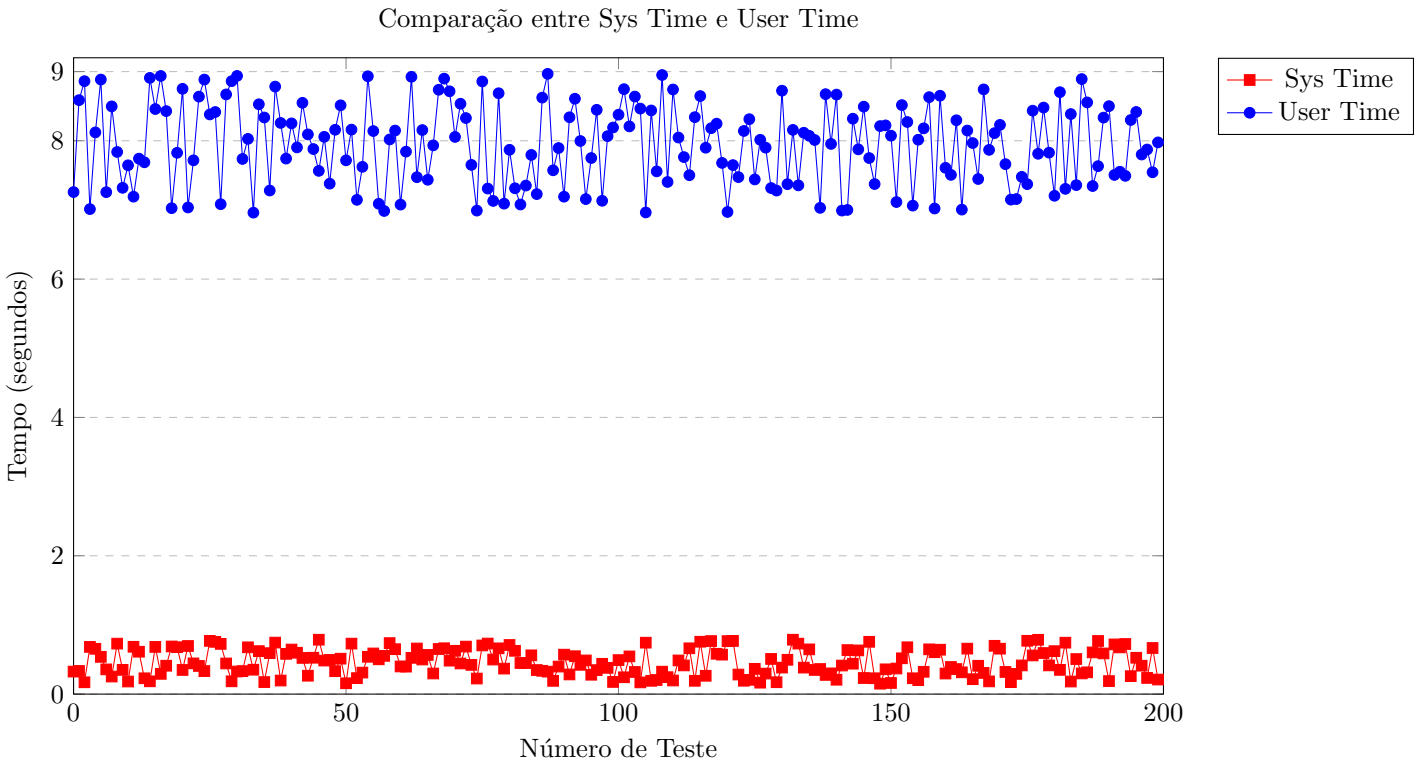
Durante o desenvolvimento do programa, foi encontrado alguns desafios, principalmente envolvendo as funções de alocação dinâmica de memória, new e delete. O maior problema com tais funções foi encontrar a linha adequada para o programa funcionar, além do fato de se valia a pena ou não utilizar vetores para o funcionamento do código.

Posteriormente, foi encontrado desafios para o funcionamento da biblioteca random, principalmente na geração aleatória

de strings, com tamanhos aleatórios entre 100 e 500 e caracteres também aleatórios.

5 Resultados





6 Endereços

Ao manipular e rodar o código "praticaDeAlocacaoDeMemoria.cpp"para exibir os endereços utilizados pelo programa, é possível perceber que não muda o endereço, sempre continua o mesmo. Por exemplo, em um caso de teste, com o código alterado para exibir somente os endereços utilizados e nada mais, todos os endereços mostrados no console foram

"0x7ffdb4f10690", que será diferente se rodar novamente, mas durante a execução do programa, se manteve a mesma e não se alterou ao longo do tempo.

7 Considerações Finais

Finalizando, conclui-se que alocação dinâmica de memória em C++ não é difícil, é muito útil, como em casos em que só se sabe a quantidade de memória necessária para rodar o programa durante a execução do programa, mas pode ser extremamente complicada em outros casos. Com isso, é notável que o gráfico da utilização de memória por tempo decorrido não é uma linha reta, mas se parece muito com uma, mostrando que em algumas vezes o programa roda mais rápido e outras um pouco mais devagar.

Percebe-se também que, no tempo de CPU, o Sys Time é muito menos utilizado que o User Time, devido ao programa necessitar menos de atividade do kernel. Por fim, é importante ressaltar que o programa não mudará o seu endereço de memória, somente quando for rodar novamente, como foi demonstrado na seção 6.