Trabalho prático 1 - Servidor de E-mails

Janderson Glauber Mendes dos Santos - Matrícula 2020054544

Departamento de Ciência da Computação - Universidade Federal de Minas Gerais (UFMG) - Belo Horizonte - MG-BRASIL.

jandersonglauber@gmail.com

1.INTRODUÇÃO

Neste trabalho, o objetivo será implementar um simulador de um servidor de emails. O sistema simulado terá suporte ao gerenciamento de contas, assim como à entrega de mensagens de um usuário para outros. O foco da simulação será verificar o funcionamento correto do sistema ao executar as diversas operações do servidor (descritas a seguir) em diferentes situações. Para a resolução desse problema a estratégia empregada será principalmente a implementação da estrutura de dados Lista.

2.MÉTODO

2.1 AMBIENTE DE DESENVOLVIMENTO:

Linguagem : C++
Compilador: g++ , contido no GNU Compiler Collection (GCC)
Sistema Operacional: Ubuntu 22.04.1 LTS

2.2 DESCRIÇÃO DA IMPLEMENTAÇÃO:

A lógica utilizada para implementar o desafio proposto consiste na implementação de três estruturas base, Servidor , Usuário e Mensagem. Servidor é uma lista que armazena uma lista de Usuários e usuários é uma lista que armazena o TAD mensagem. Cada Estrutura possui seus atributos e métodos a fim de implementar as operações base. Cadastrar ID, Remover ID, Entrega ID PRI MSG, Consulta ID.

2.2 DETALHAMENTO DA IMPLEMENTAÇÃO:

2.2.1 ORGANIZAÇÃO DOS ARQUIVOS:

A organização dos arquivos deste TP foi pensado e dividido em pastas da seguinte forma:

`raiz`: Contém as pastas que serão citadas abaixo, os arquivos txt de entrada juntamente com o arquivo `gmon.out` produzido pelo `gprof`, dedicado à análise de desempenho, os arquivos de análise de memória do `memlog` e do `analisamem`. Além disso, há o makefile para compilação.

`Obj`: Contém os arquivos da pasta `src` traduzidos a nível de máquina pelo compilador. Os arquivos dessa pasta possuem extensão `.o (object)`.

bin: Contém o arquivo executável. Esse arquivo é a união dos arquivos .o da pasta obj em um só arquivo compilado pelo GCC.

include: Contém os arquivos declaração dos TAD's utilizados no programa e que serão implementados futuramente na pasta src pelos arquivos de extensão .cpp. Tais arquivos possuem a extensão .h e possuem a declaração dos atributos e funções das classes.

- **Servidor.h:** Neste arquivo é declarado o Servidor. Esse TAD é uma estrutura de dados Lista na qual será armazenado outras listas de usuário. Há somente os atributos necessários para a implementação de uma lista e as funções necessárias para manipulá-la e para atender às exigências de operações do TP. Como principais métodos temos InsereFinal(), RemoveId(), ConsultaId(), EnviaMsg() para implementar as operações base exigidas pelo TP. As demais funções são auxiliares e padrões do tipo lista usadas para dar suporte às funções principais.

- **usuario.h:** Neste arquivo é declarado o TAD usuário. Esse TAD é uma Estrutura de dado do tipo Lista que armazena o TAD mensagem com base em sua prioridade, uma "lista de prioridade". Essa estrutura é composta pelos atributos necessários para implementar uma lista e pelo seu atributo principal Id. Como principais métodos temos InserePrioridade() e RemoveInicio(), os demais metodos InsereInicio(), InserePosicao(), Limpa() e setId() são auxiliares.

- **mensagem.h:** Neste arquivo é declarado o TAD mensagem que define as mensagens que serão inseridas na caixa de entrada do usuário. Como principais atributos desse TAD temos a prioridade e o texto da mensagem, já o tributo *prox é usado para tornar esse TAD uma célula da lista. Há apenas métodos padrões de set e get.

src: Essa pasta contém os arquivos .c que implementam as funções declaradas nos arquivos .h da pasta include. Além da lógica, também são definidas as estratégias de robustez do programa.

- mensagem.cpp: Nesse arquivo ocorre toda a implementação do TAD mensagem. Estão definidas as lógicas para os construtores, destrutores e métodos de get e set.
    - Mensagem( ) e ~Mensagem(): Tais métodos definem o destrutor e o destrutor da classe. Há dois tipos de construtores, um padrão passando o valor 0 para prioridade e vazio para texto e um no qual se passa os valores definidos para prioridade e texto. Destrutor funciona de forma padrão.
    - setPrioridade( ) : Método que define o valor de prioridade.
    - getPrioridade( ) : Método que retorna o valor de prioridade.
    - setTexto( ) : Método que define o valor de texto.
    - getTexto( ) : Método que retorna o valor de texto.
    - setPrioridade( ) : Método que define o valor de prioridade.
- usuário.cpp: Nesse arquivo ocorre a implementação de uma lista que recebe mensagem como célula. Essa lista apresenta como diferencial a capacidade de ordenação das mensagens por prioridade.
    - usuário( ) e ~usuário( ) : O construtor aloca os ponteiros para a primeira posição. O destrutor chama o método limpa desfazendo a lista e limpa o espaço de memória por meio do delete.

    - insereInicio( ) : Método que insere a célula no início da lista.

    - Posiciona( ) : Método que posiciona um ponteiro em uma posição específica da lista. Futuramente usado para a inserção de prioridade.

    - InserePosicao( ) : Método que insere a célula em uma posição específica da lista. Futuramente usado para a inserção de prioridade.

    - RemoveInicio( ) : Método que remove o primeiro elemento da lista. Esse é um método essencial aqui já que o TPI exige que após a consulta a primeira mensagem da lista sempre seja excluída.

    - Limpa( ) : Método que passa elemento a elemento limpando a lista.

    - InserePrioridade( ) : O principal método dessa estrutura. Esse método percorre a lista buscando o lugar exato para a alocação da célula com base em sua prioridade. São feitas comparações ao longo da lista , caso a mensagem possua prioridade menor ela avança a posição até ser igual ou maior que uma determinada mensagem. É importante destacar que caso a mensagem tenha prioridade igual a outra mensagem ela é alocada logo após a essa mensagem de prioridade igual.

    - SetId( ) : Define o id do usuário.

- servidor.cpp: Nesse arquivo ocorre a implementação de uma lista que recebe usuários como célula. São implementador os principais métodos referentes às operações propostas pelo TP.
    - Servidor( ) e ~Servidor() : Define o construtor e o destrutor da classe. O construtor aloca a primeira posição do ponteiro e o destrutor Limpa toda a lista e desaloca a memória.

    - Posiciona( ) : Método que dá suporte para outros métodos. Posiciona o ponteiro em um local específico da lista para que seja possível trabalhar com uma determinada posição.

- InsereFinal( ) : Insere os elementos ao final da lista. Método referente a operação de CADASTRA ID.

- RemovePosicao( ) : Remove um item de uma posição específica do vetor. Método auxiliar para que seja possível implementar a função de RemoveID. Essa função utiliza o método posiciona() para que seja possível encontrar a célula específica na lista.

- Pesquisa( ) : Método que identifica se um determinado usuário está ou não na lista. Útil para implementar a estratégia de robustez ao inserir um usuário verificando e esse já existe na lista.

- Limpa( ) : Método que passa elemento a elemento limpando a lista e desalocando memória.

- RemoveId( ) : Método que remove um usuário específico da lista. REMOVE ID
  .
- ConsultaID( ) : Método que busca por um usuário específico e ao encontra-lo remove a primeira mensagem em sua caixa de entrada. CONSULTA ID

- EnviaMsgID( ) : Método que aloca uma mensagem na caixa de entrada de um usuário específico. ENTREGA ID PRI MSG FIM

- main.cpp: principal arquivo que ordena as execuções da função do programa e garante sua conexão e funcionamento. É onde ocorre a abertura e leitura do arquivo de entrada por meio de um loop que define a alocação de cada informação presente no arquivo. Além disso, são declaradas algumas variáveis de e as chamadas das funções.

3.Análise de complexidade.

Serão descritas abaixo a complexidade de tempo e espaço das principais funções do programa:

- **mensagem.cpp:** Todas as funções desse arquivo são O(1) já que fazem apenas alocações simples.

- **usuario.cpp:**
    - InsereInicio(): O(1) pois o método não precisa percorrer a lista. É apenas uma alocação simples no início manipulando ponteiros.

    - InserePosicao(): O(1) no melhor caso já que será inserido um elemento no início ou final sem percorrer a lista e O(n) em uma determinada posição já que haverá a necessidade de percorrer a lista até encontrar o elemento específico.

    - RemoveInicio(): O(1) pois o método não precisa percorrer a lista. É apenas uma remoção simples no início manipulando ponteiros.

    - Posiciona(): Melhor caso O(1) Pior caso O(n).

    - Limpa(): O(n) pois passa por toda a lista elemento a elemento para que seja possível deslocá-lo.

    - InserePrioridade(): O(1) caso seja a primeira mensagem a ser inserida, melhor caso. O(2N) já que haverá a necessidade de percorrer a lista para que seja comparação das mensagens e mais uma vez para a inserção da mesma.

    - SetId(): O(1) , apenas comparação simples.

- **servidor.cpp:**
    - InsereFinal(): O(1) pois o método não precisa percorrer a lista. É apenas uma alocação simples no final da lista manipulando ponteiros.

    - RemovePosicao(): O(1) no melhor caso já que será inserido um elemento no início ou final sem percorrer a lista e O(n) em uma

determinada posição já que haverá a necessidade de percorrer a lista até encontrar o elemento específico.

- Pesquisa(): O(1) no melhor caso, pois o método não precisa percorrer a lista. O(n) no pior caso pois o método precisa percorrer a lista até encontrar um elemento.

- Posiciona(): Melhor caso O(1) é posicionado na primeira posição. Pior caso O(n) percorre a lista para que seja possível posicionar um elemento.

- Limpa(): O(n) pois passa por toda a lista elemento a elemento para que seja possível deslocá-lo.

- RemoveId(): O(2) caso seja a primeira mensagem a ser inserida, melhor caso. O(N^2) já que haverá a necessidade de percorrer a lista para que seja possível encontrar a posição do usuário e mais uma vez pelo método RemovePosicao().

- ConsultaID(): O(1) no melhor caso, quando o elemento desejado é o primeiro da lista. O(n) no pior caso pois passa por toda a lista elemento a elemento para que seja possível encontrar o elemento desejado.

- EnviaMsgID(): O(1) , apenas comparação simples. O($2N^2$) no pior caso pois é necessário percorrer a lista N vezes para que seja possível encontrar a posição do usuário e após isso é chamada o método InserePrioridade().

4. ESTRATÉGIAS DE ROBUSTEZ

4.1 PREVENINDO ID FORA DE ESCOPO:

De acordo com as regras do TP o id deve estar entre o seguinte limite $0 \leq ID \leq 10^6$. Pensando nisso, é implementada a estratégia de robustez no arquivo Servidor.cpp na função InsereFinal(). Existe uma condição if na qual se o id não estiver nos

limites definidos o usuário não é cadastrado. O if está vazio , literalmente não é feito nada caso o ID esteja fora de escopo. Foi implementado dessa forma para que não haja nenhuma saída que influencie na correção.Essa estratégia impede que haja ID’s errados ao longo do programa.

4.2 PREVENINDO PRIORIDADE FORA DE ESCOPO:

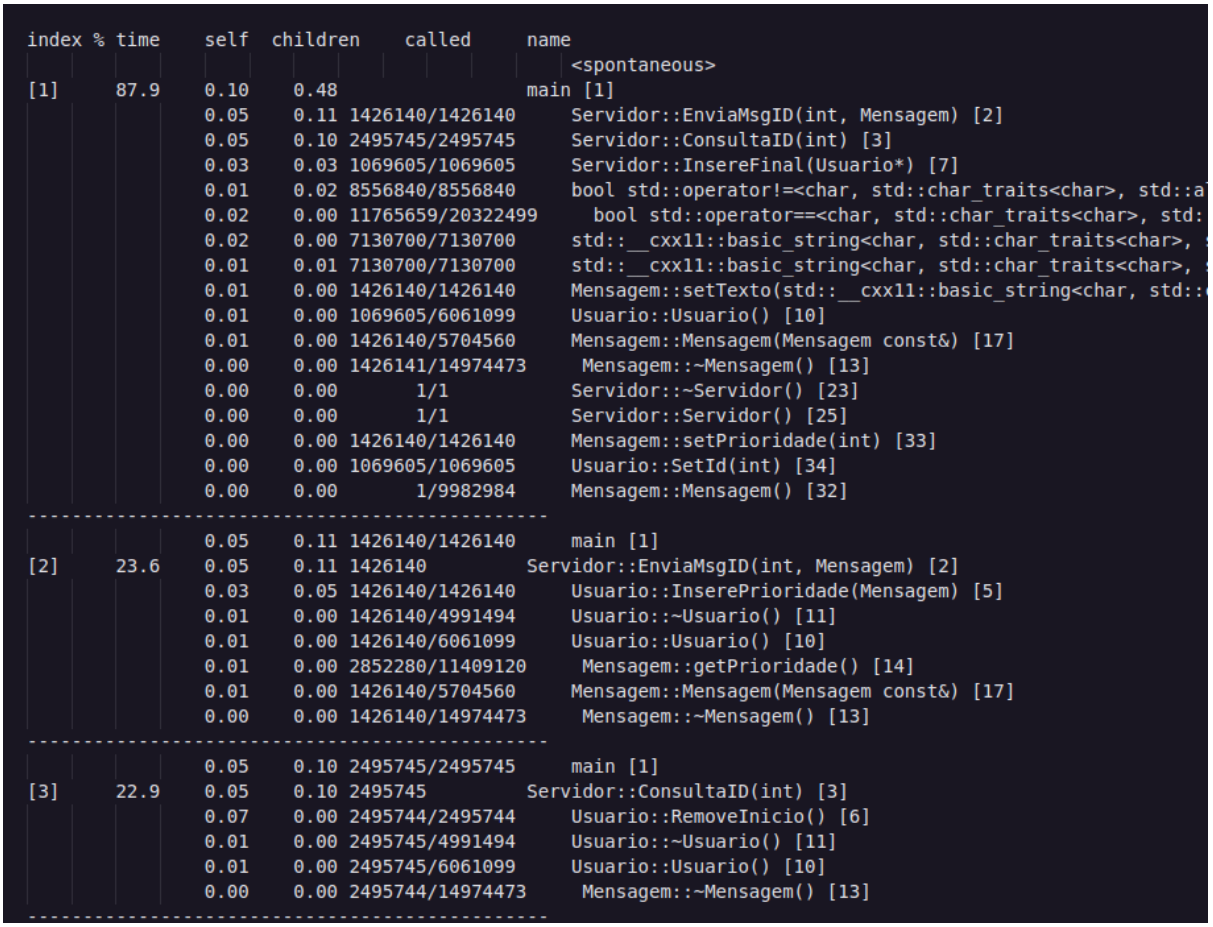
De acordo com as regras do TP a prioridade deve estar entre o seguinte limite: 0 ≤ PRIORIDADE ≤ 9. Pensando nisso, é implementada a estratégia de robustez no arquivo Servidor.cpp na função EnviaMsgID(). Existe uma condição if na qual se a prioridade não estiver nos limites definidos a mensagem não é enviada O if está vazio , literalmente não é feito nada caso a PRIORIDADE esteja fora de escopo. Foi implementado dessa forma para que não haja nenhuma saída que influencie na correção.Essa estratégia impede que haja mensagens com prioridades erradas ao longo do programa.

4.3 LISTAS VAZIAS E ELEMENTOS NÃO EXISTENTES E ELEMENTOS REPETIDOS:

Existem estratégias já estabelecidas pelo TP em caso de acesso a listas vazias e em pesquisas em que o elemento não é encontrado é printado na saída mensagens informando que elementos não foram encontrados, lista vazia, e elementos já existentes. Todos esses casos previstos pelo TP são implementados por condições através de if’s.

5. ANÁLISE EXPERIMENTAL

5.1 ANÁLISE DE TEMPO:



```
index % time    self  children    called     name
                                               <spontaneous>
[1]     87.9    0.10    0.48                 main [1]
                0.05    0.11 1426140/1426140     Servidor::EnviaMsgID(int, Mensagem) [2]
                0.05    0.10 2495745/2495745     Servidor::ConsultaID(int) [3]
                0.03    0.03 1069605/1069605     Servidor::InsereFinal(Usuario*) [7]
                0.01    0.02 8556840/8556840     bool std::operator!=<char, std::char_traits<char>, std::a
                0.02    0.00 11765659/20322499     bool std::operator==<char, std::char_traits<char>, std:
                0.02    0.00 7130700/7130700     std::__cxx11::basic_string<char, std::char_traits<char>,
                0.01    0.01 7130700/7130700     std::__cxx11::basic_string<char, std::char_traits<char>,
                0.01    0.00 1426140/1426140     Mensagem::setTexto(std::__cxx11::basic_string<char, std::
                0.01    0.00 1069605/6061099     Usuario::Usuario() [10]
                0.01    0.00 1426140/5704560     Mensagem::Mensagem(Mensagem const&) [17]
                0.00    0.00 1426141/14974473     Mensagem::~Mensagem() [13]
                0.00    0.00       1/1           Servidor::~Servidor() [23]
                0.00    0.00       1/1           Servidor::Servidor() [25]
                0.00    0.00 1426140/1426140     Mensagem::setPrioridade(int) [33]
                0.00    0.00 1069605/1069605     Usuario::SetId(int) [34]
                0.00    0.00       1/9982984     Mensagem::Mensagem() [32]
-----------------------------------------------
                0.05    0.11 1426140/1426140     main [1]
[2]     23.6    0.05    0.11 1426140         Servidor::EnviaMsgID(int, Mensagem) [2]
                0.03    0.05 1426140/1426140     Usuario::InserePrioridade(Mensagem) [5]
                0.01    0.00 1426140/4991494     Usuario::~Usuario() [11]
                0.01    0.00 1426140/6061099     Usuario::Usuario() [10]
                0.01    0.00 2852280/11409120     Mensagem::getPrioridade() [14]
                0.01    0.00 1426140/5704560     Mensagem::Mensagem(Mensagem const&) [17]
                0.00    0.00 1426140/14974473     Mensagem::~Mensagem() [13]
-----------------------------------------------
                0.05    0.10 2495745/2495745     main [1]
[3]     22.9    0.05    0.10 2495745         Servidor::ConsultaID(int) [3]
                0.07    0.00 2495744/2495744     Usuario::RemoveInicio() [6]
                0.01    0.00 2495745/4991494     Usuario::~Usuario() [11]
                0.01    0.00 2495745/6061099     Usuario::Usuario() [10]
                0.00    0.00 2495744/14974473     Mensagem::~Mensagem() [13]
-----------------------------------------------
```

Acima podemos ver a análise de desempenho em relação ao tempo do programa desenvolvido. A analise partiu do exemplo de entrada “input2.txt” disponiblizado para testes, a entrada foi repetida 4milhões de vezes para que houvesse um gasto de tempo relevante para a análise. Como podemos ver, a maior parte do tempo de execução é tomada pelas principais operações de Inserção , Consulta e Envio de mensagem. Dentre essas operações, a mais demorada e custosa é a EnviaMsgID() pois é a de maior ordem de complexidade do programa devido ao fato de percorrer 2 listas sendo O(N²). Em sequência temos ConsultaID() que também precisa percorrer a lista e possui ordem de complexidade O(N). Logo, podemos perceber na prática como a complexidade influencia no desempenho das funções.

5. CONCLUSÃO

Sumarizando, a estratégia implementada neste trabalho foi utilizar a estrutura de dados Lista como base para solução dos problemas. Temos 2 Listas aninhadas Servidor (lista1) e Usuários (lista2). Além disso, temos o TAD imagem que abstrai para o programa uma mensagem. Como diferencial desenvolvido aqui temos a lista Usuários que possui um método de inserção por prioridade.

6. BIBLIOGRAFIA

Meira, W. and Pappa, G. (2021). Slides virtuais da disciplina de estruturas de dados. Disponibilizado via moodle. Departamento de Ciência da Computação. Universidade Federal de Minas Gerais. Belo Horizonte.

[1]Analisamem Manual do Usuário Disponível em: < https://docs.google.com/document/d/1h-yeWDz2FjBVvMgI1gfXyV_ogOmwlpneKEfq4yHihoA/edi t> Acesso em: 15 de set. de 2022.

Instruções para compilação e Execução

1 – Extraia o arquivo .zip na pasta desejada.
2 – Entre na pasta do arquivo extraído pelo terminal, pasta TP1
3 – Execute o comando **"make"** no terminal para compilar os
módulos do programa
4 – Insira um arquivo.txt na raiz do programa contendo as entradas
5 – Execute o executável da pasta bin com o seguinte comando
./bin/run.out “arquivo.txt”