Trabalho Prático 1
Servidor de emails

Estrutura de Dados

Departamento de Ciência da Computação - Universidade Federal de Minas Gerais (UFMG)

Lorenzo Carneiro Magalhães
2021031505

lorenzocarneirobr@gmail.com

Belo Horizonte - MG - Brasil

1. Introdução

O trabalho prático consistiu em realizar um sistema de servidor de emails com certas propriedades e funções, de maneira que fosse escalável em termos de tempo e memória. A implementação desse sistema ocorreu com sucesso, mais detalhes do processo serão descritos a seguir.

2. Método

A implementação do servidor de emails e a caixa de emails foram feitas utilizando a estrutura de lista encadeada. Essa estrutura foi utilizada devido a sua versatilidade e custo operacional relativamente baixo de O(n). A versatilidade permitiu com que

emails fossem colocados em ordem de prioridade e caixas de mensagens fossem encontradas e removidas do sistema em qualquer posição

Também foi utilizado o sistema de classes e structs, de maneira que foram feitas as classes Email e Messagebox e os structs Node e Message. Os structs foram utilizados quando a função da estrutura era de ser os nodos de cada lista, já que não há necessidade de encapsulamento, e as classes quando a função era comportar como uma lista encadeada.

3. Análise de Complexidade

Ambas complexidades de espaço e temporal são da ordem de O(n), o que deixa o programa escalável.

Analisando temporalmente, as funções contam com, no máximo, uma estrutura de repetição que se repete no máximo pelo tamanho da lista. Ou seja, sendo n o tamanho da lista, a complexidade é O(n).

Já na análise de espaço, essa complexidade também ocorre porque a ocupação da memória ocorre em blocos de tamanhos definidos. No caso, sendo n o tamanho do bloco, a complexidade é O(n).

4. Estratégias de Robustez

Nesse programa, o sistema de robustez utilizando asserts foi utilizado para a construção inicial do código apenas. O sistema de robustez final consistiu na não execução das funções caso algo ocorresse de maneira errada, de maneira com que fosse retornado um valor identificando a falha.

Um exemplo disso ocorre ao executar a função *pop_back* quando a lista está vazia, de modo que a função seja interrompida e retorne o valor -1, indicando que não foi executado com sucesso.

Esse sistema foi utilizado porque essas saídas de erro foram utilizadas na main, que imprimia mensagens personalizadas quando os erros aconteciam.

5. Análise experimental

A análise experimental foi realizada com cautela a partir de diversos testes que testaram todas as funcionalidades do programa e foi possível concluir com clareza que o programa de fato possui complexidade de O(n).

Não foi usada a ferramenta do gprof por inconstâncias no resultado.

Os resultados dos testes podem ser visualizados na própria pasta do arquivo na execução do comando de teste (especificado na seção destinada a compilação e execução do código), mas cabe apresentar resultados parciais nessa documentação:

Cadastramento	Remoção	Entrega	Consulta	Tempo de Execução
10	10	10	10	2.74E-04
100	100	100	100	8.31E-04
100	100	10000	10000	7.84E-02
10	10	0	0	1.78E-04
100	100	0	0	6.34E-04
1	1	10	0	1.14E-04
10	10	100	0	4.87E-04
100	100	10000	0	4.18E-02

Primeiramente, é importante ressaltar que os campos abaixo das colunas Cadastramento, Remoção, Entrega e Consulta representam a quantidade de vezes da execução da função correspondente e os campos abaixo da coluna Tempo de Execução representa o tempo de execução em segundos de cada instância da aplicação dado o número de vezes de execução de cada função indicada na mesma linha.

Assim, analisando a tabela percebe-se que o tempo de execução cresce de maneira proporcional à entrada e que as funções possuem custo similares, conforme o esperado.

Uma observação importante é que os tempos de execução foram obtidos a partir de testes que não foram realizados da maneira mais limpa possível: a máquina utilizada não possuía dedicação total ao programa. Logo, o tempo de execução não representa a capacidade real da máquina.

O intuito dos testes foi mostrar a escalabilidade do programa apenas.

Também foi utilizado o valgrind para identificar possíveis *leaks* de memória, que não foram encontrados.

6. Conclusões

O trabalho prático de desenvolvimento de um servidor de emails de maneira escalável foi um sucesso. O custo computacional disso ficou proporcional ao tamanho da entrada, já que foram utilizadas duas listas, de modo a obter custo computacional de $O(n)$. Mais especificamente, cada opção da main tem custo de no máximo $2n$, que é da ordem especificada anteriormente.

7. BIBLIOGRAFIA

GNU MAKE. GNU.

8. COMPILAÇÃO E EXECUÇÃO

Para a compilação de um arquivo específico, insira o comando *make nome_do_arquivo.o* .

Para a geração do executável, digite o comando *make bin* . Esse comando irá automaticamente compilar todos os códigos e gerar um executável.

A fim de testar o código a partir dos casos teste já implementados, digite o comando *make test* .

Para o teste de performance utilizando o gprof, utilize o comando *make gprof* .

Para a análise de memória utilizando o valgrind, utilize o comando *make memcheck*. Caso use esse comando, retire a flag *-pg* da seção CFLAGS no arquivo *makefile*.

Para executar o código, utilize o comando *make run* .

Para gerar o executável e testar a performance digite *make all* .