Trabalho Prático 2 de AEDS 2

Servidor de mensagens instantâneas Dayman Novaes

**INTRODUÇÃO**

Este é um algoritmo capaz de simular um servidor de mensagens, no qual recebe em lotes mensagens trocadas por pares de pessoas (cada pessoa tem um identificador, e cada mensagem tem um identificar sequencial em relação ao par), e envia essas mensagens baseado em regras simples, como, quem enviou menos mensagens até agora tem prioridade.

O algoritmo contém três TAD’s, sendo esses:

* Mensagem
* Conversa
* Lote

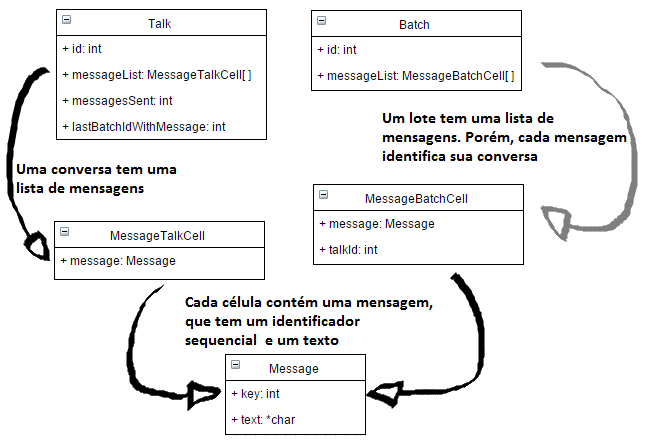
Os três serão detalhados a seguir.

Tendo como entrada cada lote, e cada mensagem com um identificador de sequência e um de conversa, o programa deverá enviar as mensagens na ordem correta e mostrar quantas mensagens cada par enviou.

**IMPLEMENTAÇÃO**

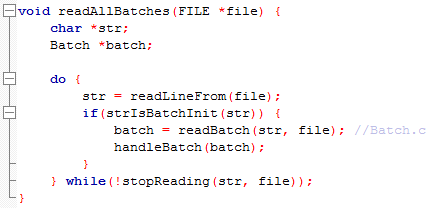
**ESTRUTURA DE DADOS**

A estrutura pode ser facilmente entendida através da imagem a seguir:



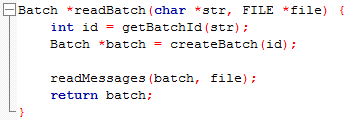
**FLUXO**

Uma das funções mais importantes para entender o fluxo do código é esta:



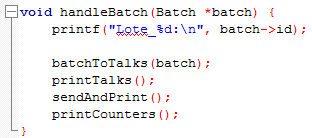
Basicamente, o que ela faz (e com o auxílio de outras cinco funções), é pegar a entrada referente a um lote e transformar isso para a struct Lote referente: batch = readBatch(str, file); e quando isso está feito, desencadeia todo o processo de imprimir as mensagens e enviá-las: handleBatch(batch);.

A função readBatch também é simples:

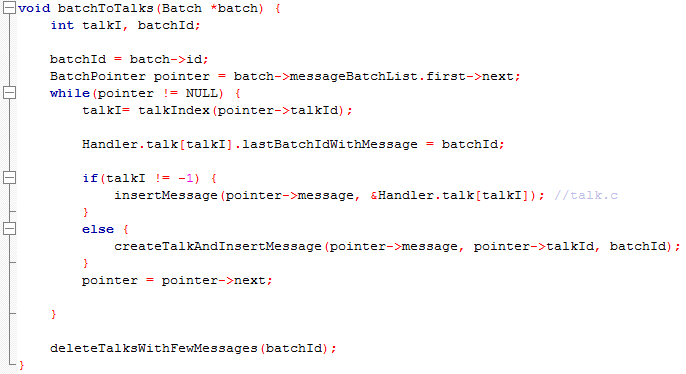


E como o auxílio de outras três funções, cria o vetor Lote, define seu identificador, e depois lê todas as mensagens readMessages(batch, file); e inclui na lista de mensagens do lote.

A função handleBatch também é muito importante, apesar de simples, pois é ela que desencadeia vários outros processos que serão responsáveis de transferir as mensagens do lote para as conversas, imprimir tudo e enviar da forma correta.



A função batchToTalks é interessante também:



Pois para cada mensagem, ela verifica se já existe sua conversa respectiva, se sim, apenas insere a mensagem nesta conversa. Se não, cria a conversa, e depois insere. Tudo isso é feito com as funções auxiliares **insertMessage** e **createTalkAndInsertMessage**. Depois de tudo, baseado no parâmetro **K**, as conversas são desalocadas: deleteTalksWithFewMessages(batchId);.

**ANÁLISE DE COMPLEXIDADE**

O programa tem como ordem de complexidade **O()**, baseado na função deleteTalksWithFewMessages, que percorre todas as conversas para achar as que devem ser deletadas, e quando acha alguma, percorre todo o array de novo para realizar o shift de todas as conversas subsequentes.

**CONCLUSÃO**

Obtive sucesso na implementação do trabalho ao obter os resultados esperados. A principal dificuldade não foi mais modularizar o código, visto que aprendi bastante com o primeiro TP. A dificuldade neste foi a alocação de memória dinâmica, pois ao lidar com várias listas diferentes, e como array de structs, e com manipulação e cópia de texto, obtive vários problemas com falha de segmentação na memória. Porém, no final consegui consertar todos e obtive um resultado satisfatório.