INF01151 –SISTEMAS OPERACIONAIS II N
SEMESTRE 2020/1
RELATÓRIO TRABALHO PRÁTICO PARTE 1: THREADS, SINCRONIZAÇÃO E COMUNICAÇÃO
Grupo: Andrews L. Rodrigues, Arthur K. Ferreira, Felipe Flores, Matheus F. Kovaleski

---

Ambiente de Testes

Ubuntu 20.04.1 LTS - Intel Core i5-4690 - 8GB DDR3 - GCC 9
Ubuntu 18.10 - AMD C-50 - 1.56GB DDR3 - GCC 8
Ubuntu 18.04.5 LTS -AMD® Ryzen 3 2200g -5,58346 GB DDR4 - G++7

Arquitetura

O grupo resolveu seguir a arquitetura sugerida pelo professor. No servidor, três componentes: o Gerenciador de Comunicação (Server), o Gerenciador de Grupos e o Gerenciador de Mensagens (Server). Do lado do Cliente, o Gerenciador de Comunicação (Cliente), Gerenciador de Mensagens (Cliente) e a Interface.
As responsabilidades de cada componente são as seguintes:

- **Gerenciador de Mensagens - Server (GMS):**

Sua função é gerenciar uma conversa para um grupo, uma instância de GMS por grupo.
A conversa é mantida em um arquivo para persistência e em um vetor em memória para rápido acesso.
Possui uma função para escrita de uma mensagem e outra para ler as últimas N mensagens. Na escrita, escreve a nova mensagem no arquivo e no vetor, na leitura, lê do vetor.
Ao ser criado, caso o arquivo do grupo já exista, reconstrói a conversa no vetor.
Guarda a conversa em plain text para facilitar a debugação.
Após escrever no arquivo, chama a função do GCS para mandar a mensagem para todos do grupo.
Detalhes sobre sincronização na seção de Sincronização.

- **Gerenciador de Grupos (GG)**

Responsável pela relação de um grupo para um instância de GMS. Cria uma nova instância de GMS na primeira mensagem para um grupo. Não possui nenhum tipo de persistência no caso do server cair, pois a persistência está garantida pelo GMS. Serve como um intermediário entre o GMS e o GCS.
Detalhes sobre sincronização na seção de Sincronização.

- **Gerenciador de Comunicação - Server (GCS)**

Responsável pela troca de mensagens entre o server e o app. Quando o server inicia é criado um socket para que a comunicação entre as duas partes seja estabelecida, após isso o sistema dá um listen no socket para ver quem está tentando se comunicar com ele. Quando uma conexão é aceita, cria-se um thread para lidar com a conexão, primeiro é recebido uma mensagem de cadastro onde o app se registra passando nome e grupo. O socket é guardado em um mapa que relaciona os grupos com os dispositivos conectados a eles, logo após isso a thread fica em loop escutado as mensagens que o usuário mandar.
Também existe um método responsável pelo replicação das mensagens para todos os usuários, onde é pego cada socket cadastrados no grupo e envia a mensagem, esse método é chamado pelo GMS após a escrita no arquivo.

- **Gerenciador de Comunicação - Cliente (GCC)**

Responsável por estabelecer a conexão com o servidor e realizar a troca de mensagens com ele. Após se conectar, se registra e cria uma thread que vai ficar em loop escutando mensagens que chegam do servidor.
O envio de mensagens para o servidor é feito por outra thread separada.

- **Gerenciador de Mensagens - Cliente (GMC)**

Intermediário entre o GCC e a Interface. Recebe da interface o texto escrito pelo usuário, monta em uma estrutura com os metadados e manda para o GCC. No sentido contrário, recebe a estrutura do GCC e já formata para o texto que será imprimido na tela pela interface.

- **Interface**

Desenha na tela a interface, consiste de uma box para mostrar as mensagens e uma box para o input do usuário. Possui um loop que fica esperando um input do usuário, após cria uma thread enviando para o GMC o texto inserido. A mensagem que chega do servidor para ser imprimida na tela vem em outra thread.
A interface é feita com a biblioteca ncurses.

- **Componentes Auxiliares**

Validation: faz a validação dos parâmetros de entrada do app

Atendendo a Múltiplos Clientes e Comunicação

Do lado do server é feito um loop infinito que fica aceitando uma conexão em cada ciclo, para cada nova conexão é criada uma thread que é responsável pelo cadastro do cliente no servidor e logo após fica em loop para que seja possível ficar escutando todas as mensagens que esse cliente mandar. Quando a mensagem é recebida ela é escrita no arquivo do server do grupo respectivo, após a escrita da mensagem no arquivo ela é mandada para todos os usuários que fazem parte daquele grupo e só após essa última etapa é aceita uma nova mensagem no arquivo.

No lado do app é criada uma thread para ficar esperando mensagens do servidor.

Sincronização no Acesso a Dados

GMServer: Uma das áreas mais críticas de sincronização é na hora de escrever ou ler mensagens no arquivo do grupo no server. Podemos ter múltiplas threads escrevendo e múltiplas threads lendo as mensagens, isso causaria diversos problemas de condição de corrida e erros tentando abrir o arquivo várias vezes. Para evitarmos isso, a primeira ação foi adicionar uma cópia da conversa em vetor de mensagens em memória também, isso permite que a leitura não tenha que acessar o arquivo. Com isso, nosso problema se resume ao problema de leitores e escritores, usamos um mutex e colocamos um shared lock para leitura do vetor da conversa e um unique lock para escrita no arquivo e no vetor (são escritos ao mesmo tempo).

GGServer: Ele possui um mapa para guardar todos os GMServers, na primeira mensagem para um grupo, ele cria o GMServer para o grupo. Logo, ele precisa testar se existe, e se não existe, insere no mapa um novo GMServer. Isso pode causar uma condição de corrida, para resolver, usamos um mutex nesse trecho de acesso e criação.

Dificuldades

Uma das primeiras dificuldades foi na definição da arquitetura e o entendimento geral do programa, depois de algumas discussões estava difícil encaixar como

todas as funcionalidades iriam funcionar junto. Resolvemos então focar no mais básico e a partir daí, construir em cima. O mais básico seria as trocas de mensagens, tentamos não nos preocupar muito com condições de corrida, sincronização de dados e threads em um primeiro momento. Após implementar esse básico, analisamos o código de novo, identificamos possíveis problemas de acesso simultâneo e condição de corrida, e tomamos medidas para solucioná-los.

A parte mais difícil de implementar foi sem dúvida os dois gerentes de comunicação, onde era o maior foco do trabalho. Após tentarmos formas diferentes de como o server poderia se comunicar com vários usuários, chegamos em uma solução. Um loop principal que vai aceitando novas conexões e lançando uma thread que fica em loop escutando o cliente conectado.

O trabalho acabou faltando algumas funcionalidades, não por dificuldades mas por falta de tempo. Então, pode-se dizer que uma das dificuldades foi a falta de tempo, falta de um maior entendimento da arquitetura por todos os integrantes do grupo e a divisão de tarefas.

Ficou faltando no trabalho as seguintes funcionalidades: recuperação das últimas N mensagens quando o usuário entra em um grupo, lidar com desconexões do lado dos clientes, número máximo de dispositivos por usuário.