Trabalho de Sistemas Distribuídos e Concorrentes:

**A Comunicação:**

A comunicação entre o server e o cliente será por sockets, sendo que UDP (User Datagram Protocol) foi o protocolo escolhido, pois diferente do TCP (Transmission Control Protocol) ele não exige que a comunicação permaneça ativa, facilitando a escrita do código, e como a confiabilidade e ordenação dos pacotes não é importante, foi o escolhido.

**Código do Servidor:**

VARIÁVEIS IMPORTANTES -

*time\_delay*: tempo que será gasto pelo processo dentro da região crítica.

*client\_buffer*: todo request recebido dos processos vai ser armazenado aqui, pode haver mais de um request de um mesmo processo, todos os requests são armazenados em fila (FIFO).

*clientes\_lock*: variável para garantir a exclusão mútua no client\_buffer

*server*: criação do socket do servidor com formato de endereço IPv4 (AF\_INET) e protocolo de transporte UDP (SOCK\_DGRAM)

*server\_address*: domínio e porta nos quais o servidor será amarrado, ‘localhost’ (127.0.0.1) e ‘7777’ respectivamente.

FUNÇÕES -

*main*: amarra o servidor a um endereço, cria as threads e as inicia.

*receiveMessage*: Recebe os requests por parte do cliente e os armazena dentro da list ‘client\_buffer’ (buffer) , há uma thread rodando essa função em loop. Após o request ser armazenado no buffer, é enviado uma mensagem de confirmação que o request foi feito.

*enterCriticalRegion*: região crítica do server, garante a exclusão mútua, quando o processo entra dentro dela, é enviado a mensagem de que o grant foi realizado. Há uma thread que executa essa função em loop, a ordem dos requests é obedecida, ou seja, o primeiro a solicitar é o primeiro a ser atendido, somente um processo por ver e o próximo processo só acessa a região quando o atual faz o release.

Toda vez que um processo acessa a região crítica ele escreve em modo append no arquivo ‘resultado.txt’ a hora atual do sistema com a sua identificação.

*sendMessage(message, address)*: Envia uma mensagem para o processo solicitado.

*deleteClient*: remove o processo da list ‘client\_buffer’ (buffer) e envia a mensagem de release.

*terminal*: função para receber um input. Sendo 1 para imprimir os atuais requests, 2 imprimir quanta vezes cada processo entrou dentro da região crítica e 3 para encerrar o server.

OBS1: sempre que for feita uma escrita, leitura, ou remoção dentro da variável client\_buffer, há a checagem da variável clients\_lock para permitir apenas uma thread acessando o buffer e não haver a condição de corrida.

**Código dos Processos:**

VARIÁVEIS

*num\_of\_procs*: número de processos.

*num\_of\_reqs*: número de requisições de cada processo.

*total\_of\_reqs*: será num\_of\_procs\*num\_of\_reqs, ou seja, o número total de solicitações ao servidor.

*client*: criação do socket cliente com as mesmas especificações do socket do server.

*server\_adress*: endereço para onde o cliente enviará as mensagens quando for necessário (precisa ser exatamente igual ao endereço em que o server está amarrado).

FUNÇÕES

*main*: cria e inicia as threads.

*sendMessage(process\_n)*: mensagem é enviada para o servidor por um dos processos. É executado em loop por uma thread.

*receiveMessage*: recebe as mensagens do servidor, podendo ser mensagem de que o REQUEST foi confirmado, ou um GRANT foi realizado e o processo acessou a região crítica, ou de um RELEASE, quando o processo encerra seu trabalho na região crítica. É executado em loop por uma thread.

OBS2: Os requests dos processos são armazenados em uma list, com um número fixo para cada um deles executar. Essa list é então embaralhada e enviada para o servidor.