Especificação do Projeto Pratico 1

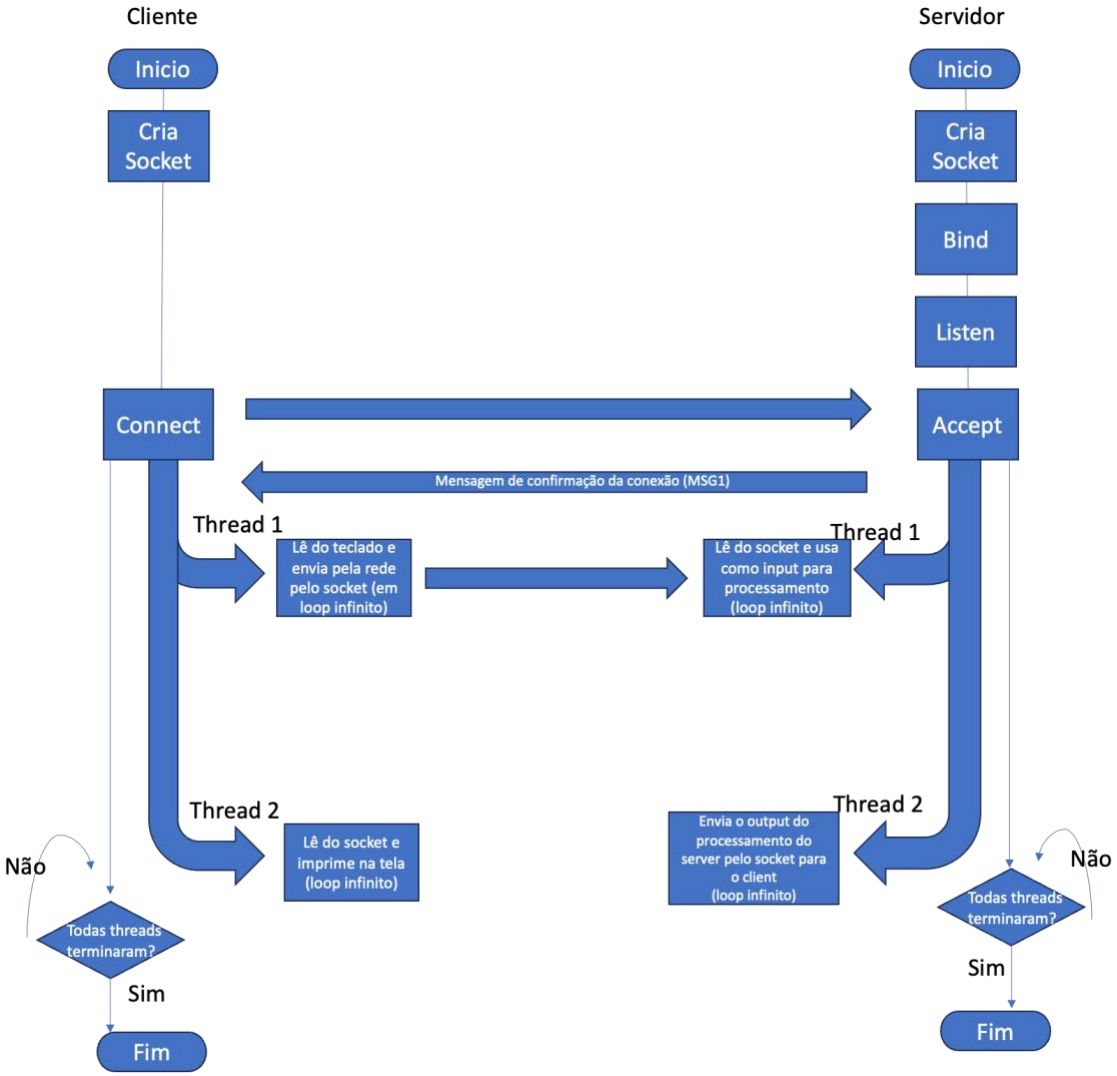
Objetivo

O projeto prático 1 busca criar conhecimentos sobre o uso das bibliotecas de Sockets para desenvolvimento de aplicações em rede no modelo client/server.
Utilizar conhecimentos e habilidades em:

- Uso de plataformas de desenvolvimento;
- Uso de threads;
- Memória compartilhada e estruturas de dados;

Todos os temas fazem uso dos mesmos conceitos, mudando de formas simples os dados e as interações, porém, sempre implementando comunicação bidirecional assíncrona na rede através da utilização de threads.

Diagrama de fluxos



Detalhamentos para os temas

Fase 1: Inicialmente, os alunos deverão implementar para um usuário remoto somente.

Monitor do sistema
O monitor do sistema deverá implementar um cliente que possa monitorar a performance do sistema operacional remoto (do servidor) remotamente: utilização de CPU e memória.
O usuário, através do cliente da aplicação, iniciará a conexão ao servidor, e receberá, de imediato, uma mensagem no seguinte formato "<HORARIO>: CONECTADO!!" e um Menu com os monitores disponíveis e sintaxe de uso (no diagrama, MSG1).

Duas threads então serão criadas, tanto no server quanto no cliente, para manipulação da conexão através do handle que identifica a conexão.

No cliente, a thread 1 receberá via teclado o comando do usuário, por exemplo, "CPU-5<ENTER>", e enviará este texto ao server pela conexão. O server receberá esta mensagem pela conexão na thread 1 do diagrama, que é responsável por um loop de leitura do socket, dará início a uma thread que executará o monitor solicitado com a periodicidade solicitada, e retornará o output periodicamente através de uma thread similar à 2 do diagrama, em loop infinito até que o usuário mande este monitor ser interrompido (o que terminará a thread do monitor, seja enviando uma mensagem ou através de uma flag em memória compartilhada – use a criatividade). O usuário poderá ainda solicitar outros monitores, que seguirão o mesmo processo com criação de threads secundárias para cada um, como descrito no exemplo acima.

O cliente, em sua thread 2 em loop infinito recebe dados da conexão e imprime na tela.

Comandos a serem implementados (e interpretados pelo server) devem ser, no mínimo:
- CPU
- memoria
- Quit (termina thread de monitoração remota)
- Exit (termina todas as threads e sai).

Chat Multiusuário
O chat multiusuário implementará uma sala de bate-papo onde vários usuários poderão se comunicar enviando mensagens públicas a todos os usuários. Nesta primeira atividade, teremos somente um usuário, portanto, o que ele enviar

O usuário, através do cliente da aplicação, iniciará a conexão ao servidor, e receberá, de imediato, uma mensagem no seguinte formato "<HORARIO>: CONECTADO!!" (no diagrama, MSG1).

Duas threads então serão criadas, tanto no server quanto no cliente, para manipulação da conexão através do handle que identifica a conexão.

O server, através da thread 2, enviará ao cliente, a cada minuto, a data e horário, independente do chat estar em idle ou de usuários estarem falando pelo canal. O cliente, por sua vez, receberá esta informação através de sua thread 2, e imprimirá na tela a informação.

O cliente, por sua thread 1, ficará esperando o usuário digitar comandos, que poderão ser:
MENSAGEM A SER ENVIADA
:nome <NOME> -> Usado para mudar o nome do usuário
:quit -> Usado para sair do aplicativo

A thread 1 enviará estes comandos ao servidor, que os receberá pela thread 1 em seu lado.
O servidor, em sua thread 1, receberá estes comandos e os armazenará em uma estrutura de dados em memória compartilhada, e voltará a esperar novos dados vindo da rede.
A thread 2 do servidor, por sua vez, fará periodicamente a varredura da área de memória compartilhada e executará a ação solicitada, seja ela definir o nome do usuário, enviar a mensagem a todos, ou desconectar um usuário.

Explicações:
1)caso o texto digitado iniciar com : (dois pontos), será interpretado como comando; caso não inicie com dois pontos, será uma mensagem a ser enviada a todos os usuários.
2) caso o usuário não defina seu nome, será atribuído automaticamente "seu IP":"porta do cliente"
3) Na tela dos usuários que recebem a mensagem, ela será formatada como:
NOME_DO_USUARIO (horário): MENSAGEM
4) Para o usuário que enviou a mensagem, ele receberá um eco sendo "Voce digitou: MENSAGEM".

Loteria
O sistema de loteria implementará uma aplicação cliente/server que faz sorteio de números para os usuários apostarem.

O usuário, através do cliente da aplicação, iniciará a conexão ao servidor, e receberá, de imediato, uma mensagem no seguinte formato "<HORARIO>: CONECTADO!!" (no diagrama, MSG1).

Duas threads então serão criadas, tanto no server quanto no cliente, para manipulação da conexão através do handle que identifica a conexão.

O usuário informará via teclado os parâmetros da loteria, sendo:
:inicio <NUMERO>
:fim <NUMERO>
:qtd <NUMERO>
APOSTA DE NUMEROS (separados por espaços)

Sendo que os dados iniciados por dois pontos (:) serão tratados como comandos para configurar a loteria, caso sejam números e espaços, será considerado como aposta. O server, por sua vez, interpretará esses dados para configurar a loteria ou fazer a aposta do cliente. Caso a loteria não seja configurada no inicio da execução, ela considerará como de 0 a 100, 5 numeros sorteados, porém esta configuração poderá ocorrer a qualquer momento, e valerá para a próxima aposta.

A thread 1 do cliente aguardará o usuário digitar os comandos ou numeros, e ao fazê-lo, enviará via conexão de rede ao server e voltará a aguardar que o usuário digite outros numeros infinitamente, até o momento do sorteio.
A thread 2 do cliente estará aguardando receber dados do server, que os imprimirá tela quando recebidos, e voltará a aguardar novos dados.
A thread 1 do servidor receberá as apostas do usuário via conexão de rede, e armazenará estes números em uma lista, e voltará a aguardar novas apostas.
A thread 2 do servidor será iniciada aguardando por 1 minuto, e fazendo sorteio dos números conforme configurado pelo usuário. Ao terminar o sorteio, ela verificará se o usuário fez apostas (percorrendo a lista referenciada acima), e verificando se alguma aposta foi sorteada. Após isso, ela enviará o resultado do sorteio ao cliente, e informará quais números ele acertou. A lista será então zerada e um novo ciclo será iniciado.

Fase 2: Implementação de multi-client: As atualizações abaixo são válidas para todos os temas

Cliente:
Deverá implementar um comando solicitando a desconexão e término da execução. O server por sua vez terminará a conexão TCP, e o cliente terminará sua execução.

Server:
- Deverá fechar a conexão TCP quando o cliente assim solicitar, e desalocar os dados daquele cliente.

-O server deverá lidar com múltiplos clientes simultaneamente que deverão ser tratados totalmente independentes. Dica: Qualquer informação ou parâmetro do serviço ao cliente deverá ser instanciado dentro das threads, que deverão ter seus handlers armazenados em listas para permitir gerenciamento das conexões pela linha principal de processamento do server, a menos que será uma estrutura de memória que seja válida para todas as threads, que neste caso, deverá ser criada como memória compartilhada.

-O server deverá limitar a quantidade de clientes através de um parâmetro passado por linha de comando ao ser executado.

-Quando um cliente desconecta, deverá ser liberada uma posição no numero de clientes disponíveis.

- Se um cliente tenta se conectar e o limite já foi atingido, o server deverá retornar uma mensagem informando isso ao invés de "<HORARIO>: CONECTADO!!" **e desconectar a conexão.**

Dica: O método listen() no socket do servidor **não causa bloqueio** na execução. Ele apenas coloca o socket em modo passivo, ou seja, preparado para aceitar conexões.

Quem realmente bloqueia a execução é o método accept(), que **aguarda até que um cliente tente se conectar**. Quando a execução continua após o accept(), isso indica que um cliente foi conectado com sucesso.

Nesse momento, o servidor deve **criar uma thread de trabalho (working thread)** para lidar com esse cliente, **passando a conexão estabelecida como parâmetro** para essa thread. Enquanto isso, a thread principal deve **retornar imediatamente ao** accept(), para continuar aguardando novas conexões.

A thread de trabalho será responsável por executar as funcionalidades da aplicação junto ao cliente. Se a quantidade de clientes conectados estiver dentro do limite permitido, o atendimento prossegue normalmente. Caso contrário, a thread deve informar que o limite foi excedido, encerrar a conexão com o cliente e finalizar sua execução, liberando recursos para os próximos.

Fase 3 – Controle de exceções. As atualizações abaixo são válidas para todos os temas.

Cliente:
O cliente deverá lidar com exceções, como erro de conexão e desconexão feita pelo server.

Server:
- Deverá lidar com exceções, como erro de conexão e desconexão feita pelo cliente sem a devida solicitação de desconexão.

- O server deverá implementar todas as funcionalidades previstas inicialmente pelo projeto, para todos os clientes, simultaneamente, com isolamento total de dados e conexões.

- O server deverá rodar continuamente, recebendo novas conexões quando as antigas sejam desconectadas, dando a manutenção correta nos contadores de conexões simultâneas.

- É esperado que nem o cliente nem o server apresente nenhum erro ou warning gerado pelo interpretador ou ambiente de runtime em situação nenhuma.