Lab02 – Construindo aplicações distribuídas usando sockets TCP/UDP

Disciplina: Programação de sistemas paralelos e distribuídos
Professor: Fernando W Cruz Turma A

Aluno: Felipe Boccardi Silva Agustini Matrícula: 180119818
Aluno: Giovanna Borges Bottino Matrícula: 170011267

Introdução:

O objetivo desse experimento é compreender as características inerentes à construção de aplicações distribuídas, incluindo passagem de parâmetros, envolvendo módulos cliente e servidor usando sockets TCP/UDP.

Descrição da solução:

O git pode ser encontrado no seguinte link:
<https://github.com/giovannabbottino/pspd_unb>
Nossas contribuições e reuniões podem ser observadas pela contribuição no git em:
<https://github.com/giovannabbottino/pspd_unb/commits/main>.

Para avaliar nosso projeto recomendo uma leitura do nosso server.c e do client.c que tentamos deixar comentado para o entendimento. Além disso, também recomendo seguir o README enviado.

Para atender os objetivos do experimento, nós optamos por uma implementação TCP contendo protocolo em modo simplex, diálogo ponto-a-ponto, stop-and-wait e orientado à conexão.

Decidimos pelo protocolo TCP por ele ser considerado o mais confiável. Enquanto o UDP não garante a entrega, o TCP garante a entrega e a integridade dos dados pacote

Para a solução de um servidor nossa maior dificuldade foi o envio de um array de float. Conseguimos solucionar através do envio do endereço de memória do array com o tamanho do tipo float multiplicado pelo tamanho do array. Usamos um arquivo *properties* para salvar o tamanho desse vetor, *VETOR*, *HOST* e *PORT.* Dessa forma, não precisamos enviar como argumento sempre. Isso também permite o cliente e o server terem acesso ao tamanho do vetor e permite alterar facilmente.

Já na solução com múltiplos servidores nossa maior dificuldade foi criar e manter a conexão paralela. Usamos de processos filhos para isso, para quantidade de servidores escolhidos diferentes existe um processo filho relacionado. Por esse motivo, na solução de múltiplos servidores também é possível usar apenas um servidor.

Diferente da solução de apenas um server, a com múltiplos server utiliza o arquivo *makefile* para conter as portas em uma lista, as portas são selecionadas através do argumento qtServer. Usamos o makefile para facilitar a chamada de 10 servidores, ou escolher entre a quantidade que pode ser de 1 a 10.

Nessa versão as portas e a quantidade de server são enviadas como argumentos para o client, enquanto o server recebe como argumento a porta a ser aberta.

Conclusão:

RPC é o serviço e protocolo que permite o código ser acionado para execução por um aplicativo remoto. Uma implementação de RPC pode ser feita com algum transporte de rede. Já o socket é apenas uma abstração de programação que pode enviar e receber dados com outro dispositivo através de um transporte de rede específico, no nosso caso TCP.

Em comparação ao trabalho 1, diferente do RPCgen que precisa ter um arquivo de definição de interface, sockets não precisam e isso facilita no envio de mensagens. Trabalhar com sockets deixa o programador com mais liberdade para achar soluções.

Abaixo temos uma tabela com o desempenho da aplicação para 1, 2, 4,6,8 e 10 workers

Número de workers	1	2	4	6	8	10
Tempo (s)	0.002	0.003	0.003	0.003	0.004	0.004

Desvio padrão: 0.000687
Média de tempo de resposta: 0.0032 s
Faixa de erro ± 0.001 s

Podemos ver um aumento do tempo de resposta conforme o aumento do número de workers, porém é um aumento esperado e nada fora do normal.

Felipe - A divisão de trabalho foi bem dividida pois usamos estratégias de programação em pares para desenvolver o código e achamos o trabalho um pouco mais fácil que o de RPC, mais pelo fato da entrega anterior ter sido feita errada. com o foco no caminho certo pudemos desenvolver sem complicações. Nota 10;

Giovanna - Percebemos que erramos bastante no trabalho 1 e resolvemos tentar recompensar nessa entrega. O trabalho foi muito bem distribuído, apenas codamos juntos. Isso permitiu que ninguém se sentisse sobrecarregado. Quando encontrávamos um problema discutimos para encontrar a solução mais simples. Nota 10.