Universidade Estadual de Campinas

MC833

Edmundo Madeira

Uso de Sockets para comunicação TCP

Autores:	*RA:*
Nathália Harumi	175188
Túlio Martins	177761

26 de Abril de 2018

1 Introdução

Neste projeto, foi explorado o uso e a eficiência da comunicação entre duas máquinas via sockets que se comunicam pelo protocolo TCP. Para que os sockets fossem implementados foram usadas bibliotecas da arpa/inet e "sys/socket.h". Para a verificação da eficiência do TCP foram medidos os tempos através de funções da biblioteca "sys/time.h". O programa utilizado para fazer tais medições é uma simulação de um banco de dados de disciplinas com acesso às informações das mesmas . O objetivo do sistema é estabelecer conexões TCP e explorar os tempos de latência das mensagens enviadas pelo protocolo entre duas máquinas em LAN e retirar conclusões a partir disso.

2 Sistema: descrição geral e casos de uso

2.1 Descrição geral do sistema

O sistema implementado consiste de dois agentes: o servidor e o cliente. O servidor contém todas as informações das disciplinas, suas estruturas de dados, e consegue suportar múltiplos clientes concorrentemente, ou seja, cria vários sockets que conseguem se conectar a porta e ouvir chamadas de vários clientes, cada socket ouvindo um cliente. Para este projeto em específico o limite foi configurado a 10 conexões simultâneas.

Já o cliente possui todas as chamadas possíveis que o servidor suporta, contendo implementações de como enviar cada requisição dependendo da funcionalidade escolhida. O cliente consiste de apenas um socket fazendo requisições que serão tratados por um único socket no lado do servidor. O cliente recebe mensagens únicas não estruturadas, ou seja, ele não está ciente da estrutura de dados e como eles são armazenados no lado do servidor.

Para mantermos a persistência nos dados foram usados arquivos. Como apenas os comentários da última aula são sobrescritos, ou seja, são alterados por clientes, apenas os mesmos estão guardados em memória permanente. Esta memória permanente reside no diretório raiz do programa do servidor e o comentário da disciplina está escrito em um arquivo com o nome da mesma disciplina.

2.2 Casos de uso

Para implementação e teste do sistema foram levados em considerações os seguintes casos de uso:

1. Um cliente pede ao sistema listar todas as disciplinas.
2. Um cliente pede para o sistema todas as informações de uma disciplina que o mesmo escolheu pelo id.
3. Um cliente pede a ementa de uma disciplina para o sistema, dado que o aluno forneceu o id da disciplina.
4. Um cliente pede todas as informações de todas as disciplinas ao sistema.
5. Um cliente professor que leciona uma dada disciplina pede requisição para escrever um comentário sobre a próxima aula desta disciplina
6. Um cliente fornece um código de uma disciplina e recebe o comentário sobre a última aula desta disciplina.

Portanto, baseado nestes casos de uso foram criadas as funcionalidades do sistema servidor e do cliente. O cliente tem estas 6 opções para escolher e o servidor provém

ao mesmo as informações de cada ua delas conforme o esperado. Os testes foram realizados entre duas máquinas diferentes em LAN.

3 Armazenamento e Estrutura de Dados do Servidor

O sistema armazena todos os dados da disciplina em uma struct definida especificamente pelo lado do servidor, do qual o cliente não necessita de conhecer. Esta struct pode ser definida conforme a figura a seguir. Como visto, a única informação guardada em memória permanente é o comentário sobre a última aula. Na struct ele é definido como um ponteiro para o arquivo de nome igual ao da disciplina.

```
typedef struct {
  char id[6];              /*Formato : MCXXX\0*/

  char titulo[LINESIZE];  /*Formato : linha de texto*/
  char ementa[TEXTSIZE];  /*Formato : texto*/
  char sala_de_aula[5];   /*Formato : CC02\0 (?)*/
  char horario[LINESIZE]; /*Formato : DIA_DA_SEMANA HH:mm a
                                  HH:mm;DIA_DA_SEMANA...\0*/

  char comentario_ultima_aula[TEXTSIZE]; /*Formato : Nome_do_arquivo*/

  char usuario[LINESIZE];
  char senha[LINESIZE];

} Disciplina;
```

Figura 1: Configuração da struct da Disciplina.

O funcionamento das informações nas structs é simples. As mesmas são armazenadas como variáveis locais com os 8 campos descritos conforma a figura. Existem 10 disciplinas armazenadas no banco, sendo elas MC833; MC102; MC536; MC750; MC358; MC458; MC558; MC658; MC346; MC886. Cada disciplina possui intrinsecamente o "id"que age como identificador para encontrar a disciplina, e os campos "titulo", "ementa", "sala_de_aula"e "horário"são strings que descrevem conforme o nome indica. O campo de "comentario_ultima_aula"indica o nome do arquivo que possui o comentário. Os últimos dois campos são strings usadas para autenticação do professor que irá escrever o comentário.

O arquivo é armazenado no mesmo diretório do servidor. Toda vez que um professor deseja escrever um comentário novo sobre a próxima aula o mesmo sobrescreve o comentário anterior. Os arquivos são gerados automaticamente caso ainda não exista nenhum comentário. Se nenhum comentário foi escrito ainda, uma mensagem é enviada para o cliente indicando que o conteúdo do comentário é "N/A".

4 Detalhes de implementação

Para a implementação deste serviço foram utilizadas mensagens de tamanhos padrões, isto é, o cliente sabe o tamanho da mensagem que espera do servidor em qualquer momento do processo. Essa tomada de decisão foi feita baseada no formato das funções "send"e "receive"das bibliotecas de programação em socket utilizadas. Como independente de como as mensagens seriam enviadas o tamanho delas deveria ser conhecida foi optado por essa abordagem.

Foram aplicados conceitos de memória permanente e concorrência. Considerando múltiplos clientes poderiam acessar o servidor simultaneamente a memória deveria ser armazenada de forma permanente para manter consistência dos dados entre os múltiplos clientes. Baseado nessa lógica que os arquivos foram introduzidos, que já fornecem permanência e trabalham bem com concorrência de acesso.

Durante a implementação foi decidido também utilizar as funções em "time.h"e "sys/time.h"para obtenção das medidas de tempo com maior precisão possível sem uso de bibliotecas fora do system. Assim, para cada uma das operações foi medido o tempo total das mesmas, e dependendo do número de "send"e "receive"utilizados pelos sockets foi realizado uma media para obtenção dos tempos das mensagens.

5 Gráficos e Interpretações

A partir das medições foram obtidos os seguintes tempos médios para cada operação como enumerada na seção de Casos de Uso pela visão do servidor:

1. $$t = (282,3 \pm 18,5)\mu s$$
2. $$t = (232,64 \pm 59,5)\mu s$$
3. $$t = (3941, \pm 859,3)\mu s$$
4. $$t = (7422,8 \pm 2283)\mu s$$
5. $$t = (10330,3 \pm 1533)\mu s$$
6. $$t = (3941, \pm 859,3)\mu s$$

É possível notar que estas médias foram afetadas por poucos pontos deviantes. A instabilidade foi claramente causada pelo acesso aos arquivos e mesmo aumentando o tamanho da amostra o desvio padrão das 3 últimas funções não foi satisfatório. A amostra total coletada foi de 50 medições por operação, incluindo a medição pura da conexão entre cliente e servidor.

Assim, a partir destes tempos foi possível fazer um gráfico com todos os tempos de envio de mensagens e recepções para que fosse obtida uma média de quanto tempo leva para uma mensagem sair do cliente e ir para o servidor e vice-versa. Assim o gráfico abaixo representa o tempo de envio destas mensagens e o tempo médio e o seu desvio padrão são dados por :

$$t = 154,3 \pm 42\mu s$$

que estão dentro do intervalo de confiança esperado.

Como visto nos gráficos abaixo temos dois padrões, as funções que necessitavam do acesso aos arquivos(Fig. 1), e os que não necessitam(Fig. 3).

Enquanto neste último gráfico os tempo medidos de conexão cliente e servido é mais estável, como esperado.

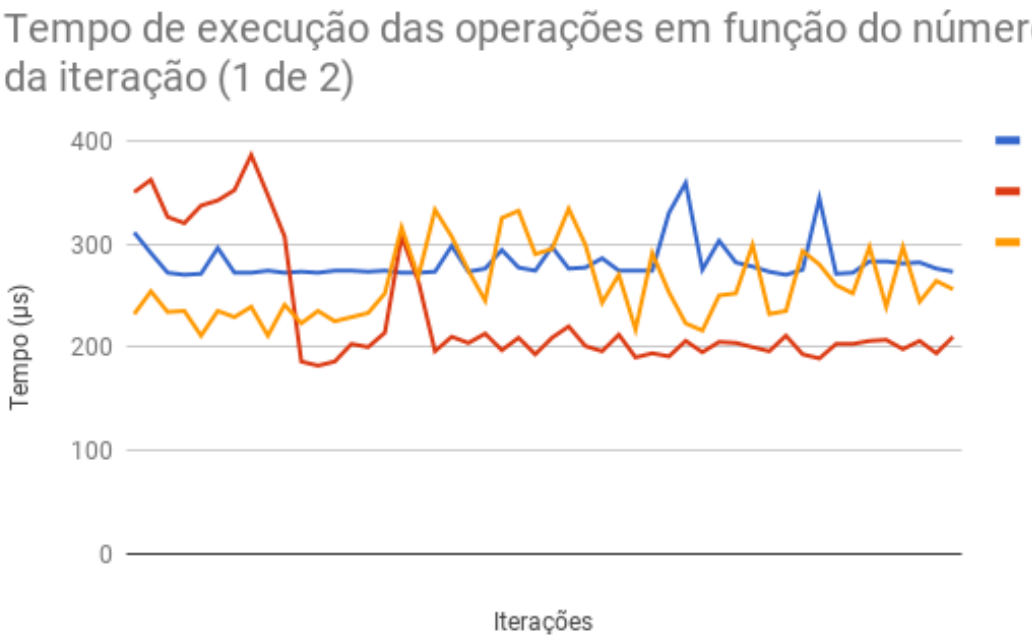

Figura 2: Gráfico do tempo(eixo y) por iteração(eixo x) das medidas obtidas das funcoes que nao necessitam de acesso aos arquivos.

Figura 3: Gráfico do tempo(eixo y) por iteração(eixo x) das medidas obtidas das fuções que necessitam de acesso aos arquivos.

Figura 4: Gráfico do tempo(eixo y) por iteração(eixo x) da conexão cliente-servidor.

6 Conclusão

A partir dos dados extraídos foi possível concluir a eficiência da execução das operações que não necessitavam o uso de arquivos. Além disso o mais importante fator observado foi o tempo da comunicação entre os sockets de cliente e servidor que resultou em 154 us. Este era um resultado esperado considerando que os testes foram feitos em LAN e serão futuramente comparados com de mensagens enviadas em outros protocolos como UDP.

Referências

[1] Stevens, R. *UNIX Network Programming. Vol. 1 "The Sockets Networking API"*. 3rd Edition, Addison-Wesley, 2003. BIMECC 005.43St47u.

[2] Beej's Guide to Network Programming, `http://beej.us/guide/bgnet/`