UNIVERSIDADE FEDERAL DO MARANHÃO
DEPARTAMENTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

Disciplina: Redes de Computadores II
Prof: Mário Meireles Teixeira

Sala de Bate-Papo

Objetivo

Implementar uma Sala de Bate-Papo (*Chat Room*) segundo o paradigma cliente-servidor, utilizando a API de sockets.

Descrição

Uma sala de bate-papo é um ambiente virtual onde múltiplos usuários podem se conectar com o intuito de trocar mensagens entre si. A sala implementada neste projeto deverá permitir comunicação síncrona, ou seja, os usuários devem estar conectados e on-line para poderem conversar.

O programa cliente deve funcionar assim: ao iniciar o cliente, o usuário deverá fornecer um usuário/senha e o hostname do servidor em cuja sala deseja entrar. O servidor deve ser capaz de suportar até 10 clientes simultâneos.

Uma vez que a conexão esteja formada, o sistema deve se comportar da seguinte forma:

- Um usuário, ao teclar [ENTER] ou pressionar [ENVIAR] em seu aplicativo cliente, irá enviar a mensagem digitada para o servidor da sala de bate-papo, que deverá replicá-la para todos os clientes conectados naquele momento;
- Alguns comandos especiais estão definidos: exit / quit para sair, neste caso o servidor deve imprimir a mensagem ‘username saiu da sala’ para todos os clientes conectados;
- O administrador do servidor também deve ser capaz de dar um comando ‘shutdown’, neste caso o servidor deve enviar uma mensagem aos clientes avisando que ele irá se desligar em 10 seg, em seguida aguardar 10 seg e então desligar-se.

No lado servidor (uma dica), as conexões dos clientes podem ser representadas por threads, ou seja, uma thread deve ser criada para cada cliente que se conectar. O cliente também deve ser implementado usando threads, pois o usuário pode estar escrevendo uma mensagem e o aplicativo simultaneamente recebendo outras.

Desafio opcional: permitir que seja possível trocar arquivos entre os clientes, ou seja, um cliente deve ser capaz de enviar uma imagem, a qual pode ser visualizada pelos outros clientes ou simplesmente feito o download por eles.

Veja no final deste documento alguns **exemplos de templates** para as aplicações cliente e servidor. Eles estão em pseudo-código em C, mas você pode escolher a linguagem de sua preferência para implementação do projeto.

O trabalho deve ser feito em equipes de, no mínimo, 2 pessoas. Interface gráfica ou Web é opcional.

Documentação

A seguinte documentação deverá ser entregue quando da demonstração do trabalho:

- Documentação externa: (texto ou slides)
    - Projeto: visão geral do funcionamento do sistema
    - Protocolo: tipos e sequência das mensagens trocadas
    - Testes: capturas de tela de uso da aplicação

- Documentação dos programas:
    - Código fonte organizado e comentado

Veja os templates do cliente e servidor na próxima página.

```
/*****************************************************************************/
/*   PROGRAM: client-template for the assignment                             */
/*                                                                           */
/*   Client creates a socket to connect to Server.                           */
/*   When the communication established, Client writes data to server        */
/*   and echoes the response from Server.                                    */
/*                                                                           */
/*   gcc -o client client.c -lpthread -D_REENTRANT                           */
/*                                                                           */
/*****************************************************************************/


int main()
{
  socket stuff (including gethostbyname(), bcopy(), socket())

  connect

  introduce the client to server by passing the nickname;

  read the echoed message;

  create a thread for reading messages;

  while(still input to keyboard)
  {
    write input to socket
  }

  close stuff ...
}




void* reader(void* arg)
{
    get FD
    while (read(FD) > 0)
    {
      print to screen what is read
    }
    close(FD)
}
```

```
/*****************************************************************************************************/
/*   PROGRAM: server-template for the assignment                                                     */
/*                                                                                                   */
/*   Using socket() to create an endpoint for communication. It returns socket descriptor            */
/*   Using bind() to bind/assign a name to an unnamed socket.                                        */
/*   Using listen() to listen for connections on a socket. Using accept() to accept                  */
/*   a connection on a socket.  It returns the descriptor for the accepted socket                    */
/*                                                                                                   */
/*   gcc -o server server.c -lpthread -D_REENTRANT                                                   */
/*                                                                                                   */
/*****************************************************************************************************/

int FDarray[MAX_CLIENT]   /* allocate as many file descriptors
                             as the number of clients  */
int counter; mutex m;
{
  socket
  bind

  listen(n)   /* n is the size of the queue that holds incoming requests
                 from clients that want to connect  */

  while(( someint = accept(socket)) > 0)
  {
    lock(m);
    FD[counter++] = someint;   /* first check for room here though */
    unlock(m);
    thr_create(bob, someint, ....)
  }

  close stuff ...

}

void bob(void* arg)    /* what does 'bob' do ? */
{
  get  fd;

  get the host name;
  read in client name;
  print a message about the new client;

  while ((read(FD, buf)) > 0)
  {
    lock(m)
    loop
      write message to each FD
    unlock(m)
  }

  lock(m);
  remove myself from FDarray
  unlock(m)
  close(FD)
  thr_exit()
}
```