

Relatório do projeto 1

Isadora Sophia Garcia Rodopoulos *
Matheus Mortatti Diamantinos †
Luiz Fernando Bittencourt‡

Abstract

O objetivo do trabalho se baseou em implementar uma estrutura de cliente e servidor que interagissem entre si.

Neste projeto, foi implementado uma estrutura de comunicação entre cliente e servidor baseado em uma conexão TCP utilizando sockets na linguagem C. Nela, o cliente pode mandar mensagens de texto para o servidor que, ao confirmar o recebimento, retorna a mesma mensagem como um *acknowledge* ao cliente.

1. client.c

O cliente precisa executar os seguintes passos:

1. Criar o socket de conexão;
2. Estabelecer conexão com o servidor;
3. Receber mensagem do usuário, mandar ao servidor e esperar resposta;
4. Se algum erro ocorrer ou o cliente fechar a aplicação, fechar a conexão.

Para isso, utilizou-se funções da library `<netdb.h>`, que nos fornece as implementações necessárias para criarmos a conexão com o servidor.

Para garantir o funcionamento da aplicação do cliente, foi estabelecida uma interface em que o usuário digita uma mensagem, e recebe o *acknowledge* do cliente como retorno. Ambas as mensagens são exibidas na tela. Além disso, caso qualquer erro tenha ocorrido no estabelecimento da conexão, um erro é exibido e o programa é imediatamente interrompido.

A seguir, serão detalhadas as funções utilizadas e seu contexto na implementação do projeto.

*RA 158018, Instituto de Computação, Universidade de Campinas, Unicamp. **Contact:** isadorasophiagr@gmail.com

†RA 156740, Instituto de Computação, Universidade de Campinas, Unicamp. **Contact:** matdiamantino@gmail.com

‡MC833, Instituto de Computação, Universidade de Campinas, Unicamp. **Contact:** bit@ic.unicamp.br

1.1. Criar o socket de conexão

Code 1. Função utilizada para criação do socket

```
int socket(int domain, int type, int protocol);
```

Code 2. Aplicação da função na implementação do projeto

```
/* create active socket */  
s = socket(AF_INET, SOCK_STREAM, 0);
```

A função acima recebe como parâmetro a família da qual o endereço pertence (no nosso caso, IPv4), o tipo de socket (SOCK_STREAM, que fornece streams de byte sequenciados, confiáveis e bidirecionais), e o protocolo a ser utilizado (0 significa que o socket vai utilizar um protocolo padrão apropriado para o tipo do socket requerido).

1.2. Estabelecer conexão com o servidor

Para estabelecer a conexão com o servidor, primeiramente buscamos o host baseado no endereço fornecido pelo usuário:

Code 3. Função utilizada para acessar o endereço do host

```
/* get ip address */  
host_address = gethostbyname(host);  
if (host_address == NULL) {  
    error("Invalid_host_name!\n");  
}
```

Esta função nos retorna o endereço do servidor na forma de `struct hostent`, que pode retornar NULL em caso de não achar o servidor.

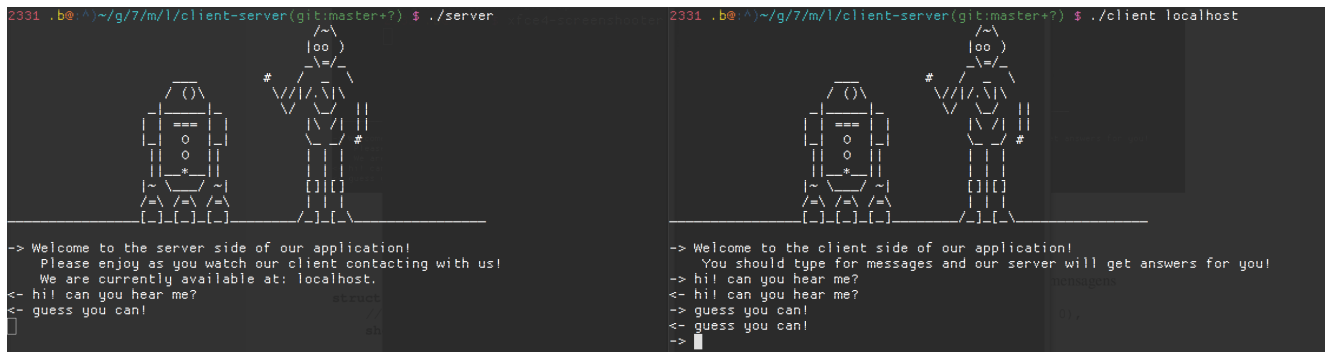
A variável `host_address` é do tipo `hostent` especificado abaixo:

Code 4. struct hostent

```
struct hostent {  
    char* h_name;  
    char** h_aliases;  
    int h_addrtype;  
    int h_length;  
    char** h_addr_list;  
    char* h_addr;  
}
```

Contudo, apenas utilizamos `char *h_addr` para acessarmos o endereço do host ao realizar a conexão TCP.

Assim, podemos utilizar a estrutura `sockaddr_in` para especificarmos a porta e o endereço do host para a conexão:



```
Code 5. struct sockaddr_in
```

```
struct sockaddr_in {  
    // AF_INET, no nosso caso  
    short                sin_family;  
  
    // porta de conexao  
    unsigned short      sin_port;  
  
    // endereco do host  
    struct in_addr       sin_addr;  
    char                sin_zero[8];  
};
```

```
Code 6. Inicialização de endereço e porta
/* initialize data address */
bzero((char*) &socket_addr,
      sizeof(socket_addr));
socket_addr.sin_family = AF_INET;
socket_addr.sin_port = htons(SERVER_PORT);
socket_addr.sin_addr =
    *(struct in_addr*)host_address->h_addr;
```

1.3. Receber mensagem do usuário, mandar ao servidor e esperar resposta

Para estabelecer a comunicação com o usuário, foi utilizada, primeiramente, a função `connect`, que permite a conexão do socket com a porta e o endereço especificados em `socket_addr`.

```
Code 7. Estabelecimento da conexão com o servidor
valid(connect(s, (struct sockaddr*) &socket_addr,
               sizeof(socket_addr)),
        "Failed...");
```

Ao iniciar a conexão, é feita a transmissão de mensagens através do socket para o servidor, com a função `send`, bastando especificar a mensagem a ser comunicada. A captura da mensagem é feita pelo input do usuário, na própria tela. Em seguida, é chamada a função `recv`, que faz o papel de receber o *acknowledge* do servidor, com a resposta a ser imprimida na tela. Caso nenhuma resposta tenha sido recebida, é assumido que a conexão foi finalizada e a aplicação é encerrada. Ao contrário, a troca de mensagens é realizada em loop.

```
Code 8. Envio e recebimento de mensagens
/* send message */
valid(send(s, buff, strlen(buff), 0),
      "Failed...");

/* receive message */
int32_t len = recv(s, buff, MAX_LINE, 0);
valid(len, "Failed...");
```

1.4. Se algum erro ocorrer ou o cliente fechar a aplicação, fechar a conexão

Para garantir que não ocorra nenhum erro conforme a aplicação está funcionando, todas as chamadas de função para qualquer API de conexão com a internet possuem verificação dos valores de retorno - e se eles fazem sentido com o contexto a qual a função foi chamada.

Além disso, foi criada uma API simples para que os erros fossem exibidos na tela de maneira uniforme e clara, automatizando o processo.

Code 9. API com verificação de erros, utilizada tanto no cliente quanto servidor

```
void error(const char* msg) {
    fprintf(stderr, "\t[ERROR]_%s", msg);
    exit(EXIT_FAILURE);
}

/* validate a status and report any errors found */
void valid(int status, const char* msg) {
    if (status == ERROR) {
        error(msg);
    }
}
```

Caso o cliente deseje fechar a conexão, a conexão é finalizada através de nosso socket, com a função `close`.

2. server.c

Semelhante ao comportamento do programa do cliente, o servidor foi dividido em:

1. Criar o socket passivo;

2. Associação do socket com o descritor;
3. Receber mensagem do cliente, exibi-las na tela e retorná-las
4. Se algum erro ocorrer ou o cliente fechar a aplicação, fechar a conexão.

Foi estabelecida uma interface bem simples, que permite que o usuário veja as mensagens que serão ecoadas na tela - as quais correspondem às mensagens que o cliente envia através da conexão com o servidor.

3. Conclusões finais

precisa mesmo?? coloquei mais por whatever.