MC833 - Relatório Projeto 1

André Muricy Santos - RA 134949
Diego Rocha - RA 135494

Funções utilizadas:

O esqueleto do projeto não foi seguido totalmente. Apesar de cada etapa ocorrer na mesma ordem, tomou-se alguma liberdade com a estrutura do código para fins de legibilidade e facilidade da nossa própria implementação. O uso de `socket_helper.c`, um arquivo auxiliar, fica evidenciado aqui.

- **`gethostbyname(char* addr):`**
  Essa função é utilizada para resolução do nome de um servidor e obtenção do endereço IP. Segundo a documentação do linux, essa resolução é feita através de DNS.

- **`bzero(&bound_address, sizeof(bound_address)):`**
  Inicializa os valores de uma struct `sockaddr_in` com valores nulos.

- **`socket(int family, int type, int flags):`**
  Cria uma socket e faz um bind nela a um valor inteiro.

- **`connect(int sockfd, struct sockaddr *sockaddr, int socklen):`**
  Tenta abrir uma conexão entre o socket identificado pelo inteiro sockfd, e o endereço contido pela struct sockaddr.

- **`bind(int sockfd, struct sockaddr *sockaddr, int socklen):`**
  Atribui um endereço, contido em sockaddr, à socket identificada por sockfd, de forma que a resolução de tal endereço direcione um cliente tentando se comunicar com ele a esse socket.

- **`listen(int listenfd, int back_log):`**
  Habilita a escuta de uma socket. Dessa forma, esse socket estará (supostamente) aceitando pedidos de conexão.

- **`accept(int sockfd, struct sockaddr *sockaddr, socklen_t *socklen):`**
  Finalmente, estabelece uma conexão que foi requisitada. A diferença entre accept e connect é que o primeiro é uma resposta ao segundo; pedidos de conexão são feitos com connect, e aceitos com accept.

Código-fonte:

server.c:

```
int main(int argc, char **argv) {
    // declaração das variáveis que serão utilizadas para a comunicação
    int listenfd, connfd;
    struct sockaddr_in socket_address;
    char buf[MAX_LINE];
    struct sockaddr_in bound_addr;
    unsigned int len;
    char recvline[MAX_LINE + 1];
    int new_s;

    // configura o valor do endereço IP do servidor
    bzero(&socket_address, sizeof(socket_address));
    socket_address.sin_family = AF_INET;
    socket_address.sin_addr.s_addr = htonl(INADDR_ANY);
    socket_address.sin_port = htons(SERVER_PORT);

    // cria o socket e armazena em listenfd
    listenfd = Socket(AF_INET, SOCK_STREAM, 0);

    // atribui o endereço ao socket
    Bind(listenfd, (struct sockaddr *) &socket_address, sizeof(socket_address));

    // habilita o recebimento de conexões por esse socket, o transformando em um socket de servidor
    Listen(listenfd, LISTENQ);

    printf("Listening for connections on port %d...\n", SERVER_PORT);

    // servidor fica aceitando novas conexões enquanto não ocorre erro,
    // porém sequencialmente (trata a comunicação atual antes de esperar pela próxima)
    for (;;) {
        // espera por uma conexão e salva as informações em connfd
        connfd = Accept(listenfd, (struct sockaddr *) &bound_addr, &len);

        do {
            // le o conteúdo enviado pelo servidor e armazena em recvline
            new_s = (int) read_line(connfd, recvline, MAX_LINE);

            // verifica se a leitura do socket falhou
            if (new_s < 0) {
                // imprime a informação do erro e termina o programa
                perror("read error");
                exit(1);
            } else if (new_s > 0) {
                strcpy(buf, recvline);

                // imprime mensagem recebida do cliente
                printf("%s:%d -> %s", inet_ntoa(bound_addr.sin_addr), ntohs(bound_addr.sin_port), buf);

                // escreve a resposta no socket do cliente
                write(connfd, buf, strlen(buf));
            }
        } while (new_s > 0);

        // finaliza a comunicação com o cliente
        close(connfd);
    }

    return 0;
}
```

client.c:

```
int main(int argc, char **argv) {
    // declara variáveis que iremos utilizar adiante para a comunicação utilizando socket
    int sockfd, n;
    char recvline[MAX_LINE + 1];
    char error[MAX_LINE + 1];
    struct sockaddr_in socket_address;
    struct sockaddr_in bound_address;
    struct hostent *host_address;
    unsigned int len;
    char buf[MAX_LINE + 1];
    int i;
    char ch;

    // valida parâmetros
    if (argc != 2) {
        // Número incorreto de parâmetros, então imprimimos como é o uso correto
        strcpy(error, "use: ");
        strcat(error, argv[0]);
        strcat(error, " <IPaddress>");
        perror(error);
        exit(1);
    }

    // resolve endereço do servidor
    host_address = gethostbyname(argv[1]);
    struct in_addr **address_list = (struct in_addr **)host_address->h_addr_list;

    printf("Server IP: %s\n", inet_ntoa(*address_list[0]));

    bzero(&bound_address, sizeof(bound_address));

    // configura o endereço do servidor para o socket
    bzero(&socket_address, sizeof(socket_address));
    socket_address.sin_family = AF_INET;
    socket_address.sin_port = htons(SERVER_PORT);
    socket_address.sin_addr = *address_list[0];


    // cria um socket e armazena em sockfd
    sockfd = Socket(AF_INET, SOCK_STREAM, 0);

    // conecta o socket ao endereço do servidor
    Connect(sockfd, (struct sockaddr *) &socket_address, sizeof(socket_address));

    len = sizeof(bound_address);
    getsockname(sockfd, (struct sockaddr *) &bound_address, &len);

    for (;;) {
        printf("-------\n");

        // le cadeia da entrada padrao
        printf("Enter the message you wish to send:\n");
        for (i = 0; (ch = getchar()) != '\n'; i++) {
            buf[i] = ch;
        }

        if (strcmp(buf, "quit") == 0) {
            break;
        }

        buf[i] = '\n';
        buf[i+1] = 0;

        // envia cadeia digitada para o servidor
        write(sockfd, buf, strlen(buf));

        // le o conteúdo enviado pelo servidor e armazena em recvline
        if ((n = read_line(sockfd, recvline, MAX_LINE)) > 0) {
            recvline[n] = 0; // adiciona um \0 como caracter terminador da string
```

```
        }

        // imprime o conteúdo recebido na saída padrão
        printf("%s\n", recvline);

        // verifica se a leitura do socket falhou
        if (n < 0) {
            // imprime a informação do erro e termina o programa
            perror("read error");
            exit(1);
        }

    }

    close(sockfd);

    // termina o programa com sucesso
    exit(0);
}
```

Abaixo, segue o código de `socket_helper.c`. Nele estão alguns wrappers e detecções de erro cujo propósito principal é a legibilidade e compartimentalização do código. Buscou-se uso de comentários claros e nomes descritivos para os parâmetros de tais funções, que contêm as chamadas para outras funções relevantes de `server.c` e `client.c`.

socket_helper.c:

```
int Socket(int family, int type, int flags) {
  int sockfd;
  if ((sockfd = socket(family, type, flags)) < 0) {
    perror("socket");
    exit(1);
  } else {
    return sockfd;
  }
}

void Connect(int sockfd, struct sockaddr *sockaddr, int socklen) {
  if (connect(sockfd, sockaddr, (socklen_t) socklen) < 0) {
       // imprime a informação do erro e terminamos o programa
       perror("connect error");
       exit(1);
  }
}

void Bind(int sockfd, struct sockaddr *sockaddr, int socklen) {
  if (bind(sockfd, sockaddr, (socklen_t) socklen) == -1) {
      // caso o ocorra erro durante a atribuição do endereço,
      // imprime o erro e terminamos o programa
      perror("bind");
      exit(1);
   }
}

void Listen(int listenfd, int back_log) {
  if (listen(listenfd, back_log) == -1) {
      // caso ocorra um erro nesse processo,
      // imprimimos o erro e terminamos o programa
      perror("listen");
      exit(1);
  }
}

int Accept(int sockfd, struct sockaddr *sockaddr, socklen_t *socklen) {
  int connfd;
```

```
  if ((connfd = accept(sockfd, sockaddr, socklen)) == -1) {
      // caso o ocorra erro ao aceitar uma conexao
      // imprime o erro e terminamos o programa
      perror("accept");
      exit(1);
  }

  return connfd;
}

// Funçao para ler uma unica linha de um file descriptor.
// Autor: Michael Kerrisk - The Linux Programming Interface
ssize_t read_line(int fd, void *buffer, size_t n) {
    ssize_t numRead;                    /* # of bytes fetched by last read() */
    size_t totRead;                     /* Total bytes read so far */
    char *buf;
    char ch;

    if (n <= 0 || buffer == NULL) {
        errno = EINVAL;
        return -1;
    }

    buf = buffer;                       /* No pointer arithmetic on "void *" */

    totRead = 0;
    for (;;) {
        numRead = read(fd, &ch, 1);

        if (numRead == -1) {
            if (errno == EINTR)         /* Interrupted --> restart read() */
                continue;
            else
                return -1;              /* Some other error */

        } else if (numRead == 0) {      /* EOF */
            if (totRead == 0)           /* No bytes read; return 0 */
                return 0;
            else                        /* Some bytes read; add '\0' */
                break;

        } else {                        /* 'numRead' must be 1 if we get here */
            if (totRead < n - 1) {      /* Discard > (n - 1) bytes */
                totRead++;
                *buf++ = ch;
            }

            if (ch == '\n')
                break;
        }
    }

    *buf = '\0';
    return totRead;
}
```

Casos de teste / exemplos de execução:

```
→ build ./client localhost
Server IP: 127.0.0.1
-------
Enter the message you wish to send:
Yummy, I love pizza!
Yummy, I love pizza!
-------
Enter the message you wish to send:
Give me moar pizza :D
Give me moar pizza :D
```

```
→ build ./server
Listening for connections on port 12345...
0.0.0.0:0 -> Yummy, I love pizza!
127.0.0.1:47702 -> Give me moar pizza :D
```

```
→ P1 netstat | grep 12345
tcp        0      0 localhost:51390         localhost:12345         ESTABLISHED
tcp        0      0 localhost:12345         localhost:51390         ESTABLISHED
```

Para os casos de teste, utilizamos localhost como parâmetro para o cliente, contando com a resolução de nome implementada em `gethostbyname()`.