Documentação

Introdução

O presente documento visa explicar a estruturação do protocolo de comunicação elaborado como solução do Trabalho Prático 2, protocolo que visa estabelecer e especificar as regras de comunicação entre duas entidade: servidor e equipamento(s).

Arquitetura

Como especificado na porposta do tp, foi adotada uma prática em que cada mensagem trocada entre cliente e servido será composta de 4 campos: Id_msg, Id_origem, Id_destino e payload. Ao receber a messagem, cada entidade identifica o a requisição/resposta e trata a mensagem de acordo com o especificado.

Servidor

Como foi pedido que os equipamentos “conversasem” entre si e que fosse feito um gerenciamento de múltiplas conexões, o servidor faz o uso de multi threads para estabelecer tais conexões simultâneas.
Cada thread ao receber uma mensagem do cliente quebra a mensagem nos quatro campos padrões (Id_msg, Id_origem, Id_destino e payload) e identifica qual a riquisição. Feito isso há um direcionamento, através de um switch, que faz o direcionamento da mensagem para uma função especifica. Essa por sua vez faz todas as verificações necessárias, de acordo com o especificado, retorna,Feito isso a resposta é enviada ao cliente, ou a thread é fechada quando há a requisição de fechamento de conexão. O banco de dados do servidor consiste em um array global.

```c
int id = 0;

while (1)
{
    memset(messageReceive, 0, BUFSZ);
    count = recv(cdata->csock, messageReceive, BUFSZ - 1, 0);

    id_msg = strtok(messageReceive, " ");
    id_ori = strtok(NULL, " ");
    id_dest = strtok(NULL, " ");

    switch (atoi(id_msg))
    {
    case 1:
        sprintf(messageToSend, "%s", add_equipment(cdata->csock, &id)); // Mensagem com novo Id
        break;
    case 2:
        sprintf(messageToSend, "%s", remove_equipment(atoi(id_ori))); // Mensagem com resposta da remoção
        break;
    case 4:
        sprintf(messageToSend, "%s", list_equipments(atoi(id_ori))); // Mensagem com resposta da remoção
        break;
    case 5:
        sprintf(messageToSend, "%s", request_information(atoi(id_ori), atoi(id_dest))); // Solicitar informação
        break;

    default:
```

Quando comecei a fazar o tp eu primeiramente montei o escopo das funções que eu precisaria, de acordo com as mensagens trocadas entre servidor e equipamento. Implementando a as funções do servidor é preciso levantar alguns pontos para melhor compreenção da lógica:

- Como o banco de dados do servidor é um array de 15 posições iniciadas com 0, ao estabelecer uma nova conexão a função add_equipament faz uma verificação se existe alguma posição diferente de 0 e caso exista retorna esse index como o id do equipamento. Além disso, o socket é guardado nessa posição do array para futuras comunicações entre o servidor e o equipamento em questão.

```c
}

char *add_equipment(int csock, int *id)
{

    char *response = (char *)malloc(BUFSZ);
    char *lista = (char *)malloc(BUFSZ);

    for (int i = 1; i < MAX; i++)
    {
        if (list_csock[i] == 0)
        {
            list_csock[i] = csock;
            *id = i;
            sprintf(response, "3 - - %.2d", *id); // Mensagem de conexão feita e Id
            printf("Equipment %.2d added \n", *id);

            for (int i = 1; i < MAX; i++)
            {
                if (list_csock[i] != 0 && i != *id)
                {
                    send(list_csock[i], response, strlen(response), 0);
                }
            }

            sprintf(lista, "%s", list_equipments(i));
            return response;
        }
    }
    sprintf(response, "%s", "7 - - 4"); // Idenditicador de limite excedido
    flag_remove = 1;
    return response;
}
```

- A função remove_equipment, por sua vez, pega o Id do equipamento e usa como index do array para retornar essa posição como 0, deixando o Id livre para futuras utilizações.

```c
char *remove_equipment(int id)
{

    char *response = (char *)malloc(BUFSZ);
    if (list_csock[id])
    {
        sprintf(response, "2 %d - -", id); // Identificador de equipamento removido
        for (int i = 1; i < MAX; i++)
        {
            if (list_csock[i] != 0 && i != id)
            {
                send(list_csock[i], response, strlen(response), 0);
            }
        }
        sprintf(response, "8 %d - 1", id); // Identificador de equipamento removido
        list_csock[id] = 0;                // Remove client da base de dados
        printf("Equipment %.2d removed \n", id);
        flag_remove = 1;
    }
    else
    {
        sprintf(response, "%s", "7 - - 1"); // Identificador de equipamento não encontrado
    }

    return response;
}
```

- A função list_equipments itera sobre toda o array do banco de dados e caso o valor da posição não seja igual a 0, o index é atribuído a uma lista que em seguida é enviada aos clientes. Fica excluído o Id do equipamento que solicitou a lista. Alḿ disso, quando exixte somente um equipamento conectado é retornado a mensagem “Lista vazia”.

```c
char *list_equipments(int id)
{
    char *response = (char *)malloc(BUFSZ);
    char *aux = (char *)malloc(BUFSZ);
    int flag_vazio = 1;

    sprintf(response, "4 - - "); // Mensagem de aquisição de identificadores

    for (int i = 1; i < MAX; i++)
    {
        if (list_csock[i] != 0 && i != id)
        {
            sprintf(aux, "%.2d ", i);
            strcat(response, aux);
            flag_vazio = 0;
        }
    }

    if (flag_vazio)
    {
        sprintf(response, "4 - - Lista vazia");
    }
    return response;
}
```

- A função request_information faz todas as verificações, como descritas no enunciado do tp, e feito isso retorna a mensagem ao equipamento solicitante, caso passe nas verificações.

```c
char *request_information(int id_ori, int id_dest)
{

    char *response = (char *)malloc(BUFSZ);
    srand((unsigned)time(NULL));

    if (list_csock[id_ori])
    {
        if (list_csock[id_dest])
        {

            float ale = 0;
            ale = rand() % 100001;
            ale = ale / 10000;

            sprintf(response, "5 %.2d %.2d -", id_ori, id_dest);
            send(list_csock[id_dest], response, strlen(response), 0);
            sprintf(response, "6 %.2d %.2d %.2f", id_ori, id_dest, ale);
            return response;
        }
        else
        {
            printf("Equipment %d not found\n", id_dest);
            sprintf(response, "%s", "7 - - 3"); // Identificador de Id target não encontrado
        }
    }
    else
    {
        printf("Equipment %d not found\n", id_ori);
        sprintf(response, "%s", "7 - - 2"); // Identificador de Id source não encontrado
    }

    return response;
}
```

Equipamento

O equipamento por sua vez foi feito para estabelecer uma única conexão com o servidor. Entretanto como temos que receber respostas do servidor e enviar requisições paralelamente, assim como no servidor foi o uso de multi threads, uma para ser mais exato. A conexão principal fica responsável por enviar as respostas do servdor e única thread fica responsável por receber as requisições ao servidor.
Ao receber um comando do terminal esse comando é transferido para uma função de mapeamento que converte esse comando em uma mensagem padrão adotada pela arquitetura (Id_msg, Id_origem, Id_destino e payload).

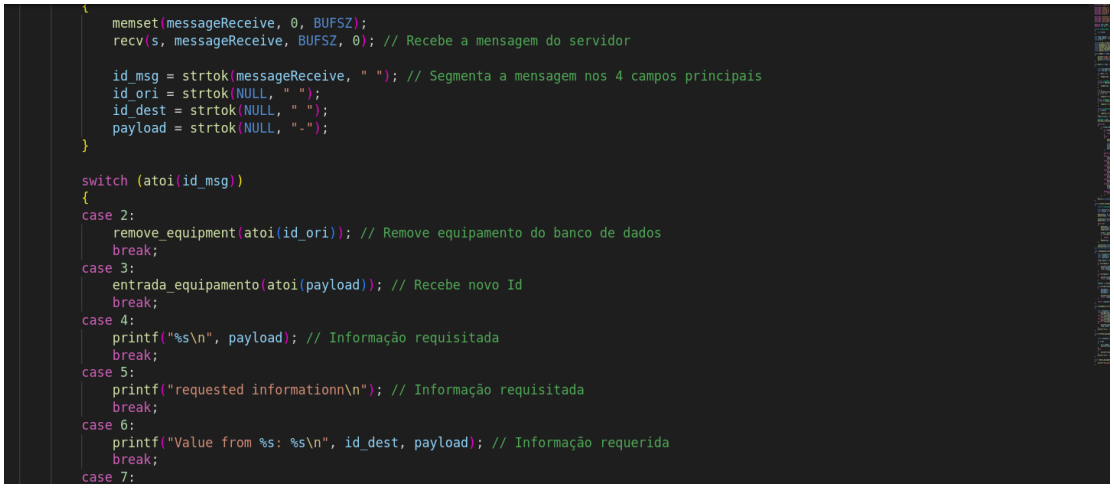
```c
char *mapeamento_mensagens(char *buf, int id, int s)
{
    char *response = (char *)malloc(BUFSZ);
    char *request = (char *)malloc(BUFSZ);
    char *equipment = (char *)malloc(BUFSZ);

    flag_request = 0;

    if (strcmp(buf, "close connection\n") == 0)
    {
        sprintf(response, "2 %.2d - -", id); // Mensagem para fechar conecção
        flag_request = 1;
        return response;
    }

    if (strcmp(buf, "list equipment\n") == 0)
    {
        sprintf(response, "4 %.2d - ", id);
        return response;
    }
                          (char [2])" "
    request = strtok(buf, " ");

    if (strcmp(request, "request") == 0)
    {
        equipment = strtok(NULL, " ");
        equipment = strtok(NULL, " ");
        equipment = strtok(NULL, "\n");

        sprintf(response, "5 %.2d %s -", id, equipment); // Mensagem de aquisição de informação
        return response;
    }

    return response;
}
```

Essa função retorna a mensagem que será enviada ao servidor.

```c
send(cdata->csock, buf, strlen(buf), 0);
memset(buf, 0, BUFSZ);

while (1)
{
    fgets(buf, BUFSZ - 1, stdin);
    sprintf(buf, "%s", mapeamento_mensagens(buf, id, cdata->csock)); // Mapeamento mensagem terminal para mensagem enviada ao servidor
    send(cdata->csock, buf, strlen(buf), 0);

    fecha = strtok(buf, " ");
    if (strcmp(fecha, "2") == 0)
    {
        break;
    }
    memset(buf, 0, BUFSZ);
}

close(cdata->csock);
pthread_exit(EXIT_SUCCESS);
```

Ao receber uma mensagem, o cliente segue a mesma lógica do servidor: ele segmenta a mensagem recebida nos quatro campos padrões e identifica qual a resposta o servidor enviou, através do Id_msg.

Após isso há um direcionamento dessa mensagem para uma função, de acordo com a mensagem, que trata a mensagem e imprime no terminal a resposta do servidor. Entretanto algumas mensagns são apenas imprimidas no terminal, sem a necessidade de uma função.



```
{
    memset(messageReceive, 0, BUFSZ);
    recv(s, messageReceive, BUFSZ, 0); // Recebe a mensagem do servidor

    id_msg = strtok(messageReceive, " "); // Segmenta a mensagem nos 4 campos principais
    id_ori = strtok(NULL, " ");
    id_dest = strtok(NULL, " ");
    payload = strtok(NULL, "-");
}

switch (atoi(id_msg))
{
case 2:
    remove_equipment(atoi(id_ori)); // Remove equipamento do banco de dados
    break;
case 3:
    entrada_equipamento(atoi(payload)); // Recebe novo Id
    break;
case 4:
    printf("%s\n", payload); // Informação requisitada
    break;
case 5:
    printf("requested informationn\n"); // Informação requisitada
    break;
case 6:
    printf("Value from %s: %s\n", id_dest, payload); // Informação requerida
    break;
case 7:
```

Problemas encontrados

Inicialmente eu não tive problemas ao construir o protocolo, pois foquei em conseguir implementar as especificações de comunicação para um cliente, levando em conta as mensagens que não necessitava de outro cliente. Após isso eu criei as funções que necessitavam de ter mais de um cliente conectado, entretanto eu criei dados falsos para testar essas funções. Feito isso eu tentei realizar o broadcast usando TCP, entretanto eu tive muito dificuldade em implementar. Mesmo quando a especificação sobre o uso do TCP mudou, quando foi permitido usar o UDP, eu ainda não conseguir implementar o broadcast. Até que junto a alguns colegas e fazendo algumas pesquisas eu consegui implementar ele.

Outro problema que tive foi que ao fechar a conexão com um equipamento, quando eu usava o mesmo terminal para criar uma nova conexão, eu me deparava com um segmentation fault. Depois de algumas análises eu consegui enxergar que o erro: dentro do meu loop eu não verifiquei se a resposta do servidor era vazia, então ao fazer a requisição daconecção eu recebia a resposta e recebia outras respostas vazias. Essas respostas vazias estavam quebrando com minha lógica de tratamento das respostas, e isso ocasionava no segmantation fault. Para resolver eu fiz uma verificação do tamanho da resposta do servidor, se fosse igual a 0 o fluxo ignorava a resposta.

Outro problema que notei é que ao dar o comando request information from em alguns momentos a mensagem não era recebida pelo servido, apesar de ser eviada pelo equipamento. Eu não consegui achar a reiza desse problema.

conclusão

Após a implementa,cão do TP1 foi possível aproveitar muitas de suas lógicas e até mesmo partes do código no O TP2. Entretanto isso não diminuiu o nível de dificuldade do TP2. Foi bastante proveitoso o tempo gasto aprendendo mais sobre o protocolo UDP, como estabelecer uma coneção usando-o, e também pesquisando sobre o métodos de implementar o broadcast usando o protocolo TCP. Apesar de eu não ter conseguido implementar em UDP valeu o conhecimento adquerido. Eu pude notar também que a segmentação das mensagem trocadas entre servidor e equipamento facilitou a estruturação do tratamento das mensagens, além de ter sido um norte no começo da implemetnação do TP. Por fim, é necessário ressaltar que apesar de trabalhoso e difícil foi gratificante implementar o TP2, apesar de suas falhas.