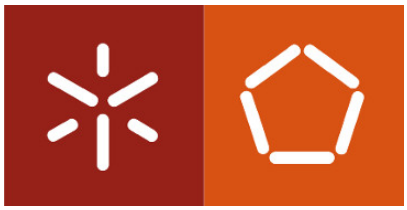


Universidade do Minho

Escola de Engenharia

Mestrado Integrado em Engenharia de Telecomunicações e Informática

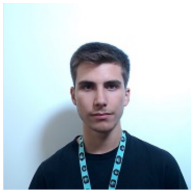
Serviços de Rede e Aplicações Multimédia

Trabalho Prático Nº2 – Sistema de Distribuição Multimédia 'Live'

Professor Dr. Bruno Alexandre Fernandes Dias

26 de junho de 2023

Grupo:



Rui Cunha - A93079
a93079@alunos.uminho.pt

Francisco Martins - A93079
a93079@alunos.uminho.pt

Conteúdo

Lista de Figuras

Lista de Tabelas

1 Introdução

Este relatório está inserido no âmbito da Unidade Curricular Serviços de Rede e Aplicações Multimédia, do 2ºsemestre do 4ºano do Mestrado Integrado em Engenharia de Telecomunicações e Informática, como resposta ao problema apresentado pelo professor. Para realizar este trabalho recorre-se as bases dos trabalho prático inicial que visa promover a familiarização dos alunos com as ferramentas utilizadas neste trabalho e ao material existente na *BlackBoard* [1] .

O objetivo deste trabalho consiste mo desenvolvimento de um sistema simples, mas completo, com elementos (fonte de informação, servidor multimédia, cliente multimédia e interface de utilizador com reprodutor multimédia) que implementem certos processos básicos.

2 Estratégias utilizadas

Aqui iremos falar nas estratégias definidas para a realização do sistema. Também iremos falar das decisões tomadas em relação aos mecanismos adotados.

2.1 Fonte de informação

O sistema inclui um programa que gera informação multimédia a um certo ritmo num determinado formato de codificação e que envia o resultado (a um certo ritmo) para o servidor multimédia. O ritmo e a Porta UDP do servidor multimédia são parâmetros recebidos no inicio do programa. A informação a gerar é o valor duma onda sinusoidal *sin(x)* com uma determinada frequência (F), amostrada a uma frequência Fa múltipla de F (Fa=N*F, N 3). Os valores F e N, são parâmetros configuráveis, no menu no inicio do programa, e determinam, indiretamente, o ritmo a que a informação é gerada. Contudo, em cada segundo, irá ser gerado um valor (Vi) de amostra a um certo ritmo através da formula:

$$Vi = int(1 + (1 + sin(2 * \pi * \frac{i}{N})) * 30$$

2.1.1 Protocolo P1

O protocolo P1 de transmissão duma fonte de informação para o servidor multimédia é encapsulado em UDP e a sua comunicação tem um papel passivo, logo a fonte de informação não sabe se o servidor multimédia esta a receber corretamente ou não. O protocolo é projetado para ser seguro, com o uso de valores especiais durante o primeiro período para sinalizar o início do programa. Vários programas podem contribuir simultaneamente para a mesma fonte, desde que usem o mesmo identificador.

O protocolo P1 tem a seguinte fisionomia:

Tabela 1: Protocolo P1.

D	i	Vi	P	F	N	M

Os parâmetros do P1 tem as seguintes funções:

- **D**: Identificador associado á fonte de informação.
- **i**: Valor aleatório entre 1 ≤ i ≤ F * N = Fa.
- **Vi**: Valor de amostra gerado.
- **P**: Período da função em que são gerados valores Vi.
- **F**: Frequência.
- **N**: Número de valores gerados por segundo.
- **M**: Valor máximo do P.

2.2 Servidor Multimédia

O Servidor Multimédia (SM) é composto por três módulos distintos:

- Módulo de Receção:Responsável por receber toda a informação de todas as fontes em uma porta fixa. Ele mantém uma lista de fontes de informação ativas e pode transmitir essa lista aos clientes para que possam escolher quais fontes desejam receber. Esse módulo implementa o protocolo P1 para a comunicação com as fontes. Na figura 1 podemos ver a estrutura utilizada para armazenar a informação da fonte.

```
typedef struct {
    char *clients_id[5];
    int F;
    int N;
    int M;
} source;

source src[5];
```

Figura 1: Estrutura utilizada para armazenar informação sobre a fonte multimedia.

- Módulo de Envio: Encarregado de enviar a informação relevante para os clientes que se subscreveram as fontes. Ele implementa o protocolo P2 para interagir com os clientes multimédia. Esse módulo lida apenas com o processo de envio da informação multimédia aos clientes usando o protocolo P2.

- Módulo de Gestão de Subscrições: Responsável pela gestão das subscrições dos clientes. Esse módulo recebe comandos dos clientes e responde de acordo com o protocolo P2. Existem quatro tipos de comandos disponíveis para os clientes: list, info(D), play(D) e stop(D).

2.2.1 Protocolo P2

O protocolo P2 é utilizado para a interação entre o Servidor Multimédia e os clientes. Ele permite a transmissão de dados dos canais para os clientes e a gestão das subscrições. Tal como o protocolo P1, o P2 é encapsulado em UDP.

- No processo de transmissão de dados do canal para um cliente, o SM recebe um PDU-1 com informação relevante, forma um PDU-2 e envia para todos os clientes subscritos ao canal. Os PDU-2 são encapsulados em datagramas UDP e contem informações adicionais, como um campo de identificação do tipo de PDU-2.

- No processo de gestão de subscrições, o SM recebe comandos dos clientes, como "list", "info", "play"e "stop", para iniciar e terminar as subscrições dos canais. O SM responde com PDU-2s específicos para cada comando, como "response-list"e "response-info". Cada comando possui um identificador único, gerado pelo cliente, que é incluído nos PDU-2s de resposta para que o cliente possa identificar o comando correspondente.

O protocolo P2 foi projetado levando em consideração a natureza assíncrona e não confirmada da comunicação, proporcionando interações eficientes entre o SM e os clientes.

2.3 Clientes

Os clientes multimédia possuem um módulo para processamento da informação para exibição consistente na interface do utilizador (UI) e gestão da interação com os utilizadores. Os clientes também possuem um modulo que permite ao utilizador, no (UI), ver a lista de canais disponíveis, escolher um para reprodução ou para parar a reprodução. O cliente também dispõe de um ID_Cliente que é enviado como argumento ao iniciar o programa.

2.3.1 Comandos do Cliente

Os clientes tem a possibilidade de introduzir quatro comandos:

- **list**: Quando recebe o comando list, o servidor responde com um "response-listIn", em que o n corresponde ao numero de fontes de informação disponíveis no servidor, e envia o D de todos os canais disponiveis.
- **info (D)**: Para o comando info (D), o servidor responde com "response-info"e os parâmetros disponíveis sobre a fonte D (D, F, N, M).
- **play (D)**: Ao receber o comando play (D), o servidor multimédia responde com um "response-play"e adiciona o cliente à lista de subscritores do canal D.
- **stop (D)**: Quando recebe stop (D), o servidor responde com "response-stop"e remove o cliente da lista de subscritores do canal D, desde que o cliente esteja subscrito ao canal.

3 Funções implementadas

Nesta secção do relatório, vamos explorar as funções em C implementadas para fazer com o sistema completo funcione como esperado.

Na função apresentada na figura 2, conseguimos observar como são enviados os PDU-1 para o servidor. Inicialmente, o Vi é enviado a 0 durante a primeira amostra, de seguida o Vi é calculado e enviado no P1 em conjunto com os outros valores.

```
while (true) {
    P = 1;
    if (index == 0) {
        for (int counter = 1; counter <= Fa; counter++) {
            int i = (rand() % Fa) + 1;

            sprintf( s: buffer,  format: "%s|%d|%d|%d|%d|%d|%d|", id_string, i, 0, P++, F, N, Fa);
            send( fd: client_fd,  buf: buffer,  n: 25,  flags: 0);

            sleep( seconds: (1 / Fa));
        }
    } else {
        for (int counter = 1; counter <= Fa; counter++) {
            double t = (double) counter / Fa;
            int i = (rand() % Fa) + 1;
            double sample = 1 + (1 + sin( x: 2 * PI * t / N)) * 30;

            sprintf( s: buffer,  format: "%s|%d|%d|%d|%d|%d|%d|", id_string, i, (int) sample, P++, F, N, Fa);
            send( fd: client_fd,  buf: buffer,  n: 25,  flags: 0);

            sleep( seconds: (1 / Fa));
        }
    }
    index++;
}
```

Figura 2: Função utilizada pela fonte de informação para enviar os P1 para o servidor.

Na figura 3, conseguimos ver a operação por detrás do servidor quando recebe o comando list. Ao receber o PDU do cliente, o servidor processa e envia ao cliente um "response-list"com o numero de canais disponiveis, separado por uma coluna. De seguida, envia ao cliente o identificador de todos os canais ativos.

```
if (strcmp(token, "list") == 0) {   // If data received is "list"

    // Lock mutex to store content
    pthread_mutex_lock(&mutex);

    sprintf( s: buffer_to_send,  format: "response-list|%d", num_sources);
    if (sendto( fd: socket,  buf: buffer_to_send,  n: 16,  flags: 0,  addr: (struct sockaddr *) &clientAddress,  addr_len: clientAddressLength) ==
        -1) {
        perror( s: "ERROR!");
        exit( status: 0);
    }

    for (size_t i = 0; i < (size_t) num_sources; i++) {
        sprintf( s: buffer_to_send,  format: "%zu. %s", (i + 1), channel_id[i]);
        if (sendto( fd: socket,  buf: buffer_to_send,  n: 20,  flags: 0,  addr: (struct sockaddr *) &clientAddress,
                    addr_len: clientAddressLength) == -1) {
            perror( s: "ERROR!");
            exit( status: 0);
        }

    }

    // Unlock mutex to store content
    pthread_mutex_unlock(&mutex);
```

Figura 3: Função utilizada no servidor para enviar ao cliente a lista de canais disponíveis.

Quando o servidor recebe o PDU "info (D)", o servidor separa o identificador (D) do resto da mensagem para poder procurar no *array channel_id* pelo index do (D). Quando obtiver o (D) o servidor envia ao cliente um "response-info"com os parâmetros (D, F, N, M) do canal, separado por uma vírgula. O processamento do PDU "info (D)"pode ser visto na figura 4.

```
} else if (strcmp(token, "info") == 0) {    // If data received is "info (D)"
    int index = 0;
    token = strtok( s: NULL,  delim: " ");

    // Lock mutex to store content
    pthread_mutex_lock(&mutex);

    for (int i = 0; i < num_sources; i++) {
        if (strcmp(channel_id[i], token) == 0) {
            index = i;
        }
    }

    sprintf( s: buffer_to_send,  format: "response-info,(%s|%d|%d|%d)", channel_id[index], src[index].F, src[index].N,
            src[index].M);
    if (sendto( fd: socket,  buf: buffer_to_send,  n: 35,  flags: 0,  addr: (struct sockaddr *) &clientAddress,
                addr_len: clientAddressLength) == -1) {
        perror( s: "ERROR!");
        exit( status: 0);
    }

    // Unlock mutex to store content
    pthread_mutex_unlock(&mutex);
```

Figura 4: Função utilizada no servidor para enviar ao cliente a informação do canal selecionado.

Nas figuras 5 e 6, podemos ver a função usada para processar o PDU "play (D)". O servidor separa o identificador (D) do resto da mensagem para poder efetuar a sua busca no *array* channel_id e envia ao cliente a mensagem "response-play"para indicar ao cliente que o pedido foi bem sucedido. Na figura 5, é apresentada a função usada para adicionar o *ID_Cliente* ao *array* clients_id da fonte selecionada para a informação multimédia ser enviada para o cliente.

```
} else if (strcmp(token, "play") == 0) {    // If data received is "play (D)"
    int index = -1;
    char * source_information;
    token = strtok( s: NULL,  delim: " ");

    // Lock mutex to store content
    pthread_mutex_lock(&mutex);

    for (int i = 0; i < num_sources; i++) {
        if (strcmp(channel_id[i], token) == 0) {
            index = i;
        }
    }

    // Unlock mutex to store content
    pthread_mutex_unlock(&mutex);

    if (index == -1) {
        sprintf( s: buffer_to_send,  format: "channel does not exists...");
        if (sendto( fd: socket,  buf: buffer_to_send,  n: 30,  flags: 0,  addr: (struct sockaddr *) &clientAddress,
                    addr_len: clientAddressLength) == -1) {
            perror( s: "ERROR!");
            exit( status: 0);
        }
        break;
    } else {
        sprintf( s: buffer_to_send,  format: "response-play");
        if (sendto( fd: socket,  buf: buffer_to_send,  n: 30,  flags: 0,  addr: (struct sockaddr *) &clientAddress,
                    addr_len: clientAddressLength) == -1) {
            perror( s: "ERROR!");
            exit( status: 0);
        }
    }

    // Lock mutex to store content
    pthread_mutex_lock(&mutex);
```

Figura 5: Função utilizada no servidor para adicionar o cliente ao *array* do canal selecionado.

```
// Lock mutex to store content
pthread_mutex_lock(&mutex);

token = strtok( s: NULL,  delim: " ");

int i = 0;
do {
    if (src[index].clients_id[i] == NULL) {
        src[index].clients_id[i] = (char *) malloc( size: 2);
        memcpy( dest: src[index].clients_id[i],  src: token,  n: sizeof(&token));
        i = num_clients;    // Leave the loop
    }
} while (i < num_clients);

num_clients++;

// Unlock mutex to store content
pthread_mutex_unlock(&mutex);
```

Figura 6: Função utilizada no servidor para adicionar o cliente ao canal selecionado.

Quando o servidor receber o PDU "stop (D)", o servidor separa o identificador (D) do resto da mensagem e verifica a existência do (D) no servidor. De seguida, efetua uma busca pelo o ID_Cliente no *array* clients_id da fonte selecionada e retira-o do *array* para anular a subscrição do canal. Quando a operação tiver sido efetuada com sucesso envia a mensagem "response-stop"ao cliente. O processamento do PDU "stop (D)"pode ser visto na figura 7

```
} else if (strcmp(token, "stop") == 0) {    // If data received is "stop (D)"
    int index = 0;
    token = strtok( s: NULL,  delim: " ");

    // Lock mutex to store content
    pthread_mutex_lock(&mutex);

    for (int i = 0; i < num_sources; i++) {
        if (strcmp(channel_id[i], token) == 0) {
            index = i;
        }
    }

    token = strtok( s: NULL,  delim: " ");

    int i = 0;
    do {
        if (strcmp(src[index].clients_id[i], token) == 0) {
            free( ptr: src[index].clients_id[i]);
            i = num_clients;    // Leave the loop
        }
    } while (i < num_clients);

    num_clients--;

    // Unlock mutex to store content
    pthread_mutex_unlock(&mutex);

    sprintf( s: buffer_to_send,  format: "response-stop");
    if (sendto( fd: socket,  buf: buffer_to_send,  n: sizeof(buffer_to_send),  flags: 0,  addr: (struct sockaddr *) &clientAddress,
                addr_len: clientAddressLength) == -1) {
        perror( s: "ERROR!");
        exit( status: 0);
    }
}
```

Figura 7: Função utilizada no servidor para retirar o cliente do *array* do canal selecionado.

4 Testes

De modo a testar o sistema multimédia, foram efetuados vários testes para verificar o funcionamento do nosso sistema final. Na figura 8, podemos ver a execução da fonte de informação multimédia com a adição dos parâmetros F e N.

```
rui@Rui:~/Desktop/Multimedia/TP2$ ./fonte 5000 canal1
Set number for F: 1
Set number for N, N > 3: 10
```

Figura 8: Teste de execução da fonte de informação multimédia.

Na figura 9, temos a execução do servidor com as *prints* que funcionam como *debug* e mostram informação recebida como os PDU provenientes do cliente e informação sobre o numero de fontes conectadas ao servidor.

```
rui@Rui:~/Desktop/Multimedia/TP2$ ./server
Server listening on port 5005...
Server listening on port 5000...
Num_sources: 1
list
play
```

Figura 9: Teste de execução do servidor.

Na figura 10, temos a execução da interface dos clientes com as opções disponíveis.

```
rui@Rui:~/Desktop/Multimedia/TP2$ ./client chico
1. Conectar ao servidor
2. Sair
Escolha uma opção: 1

1. list
2. info (D)
3. play (D)
4. stop (D)
5. Sair
Escolha uma opção:
```

Figura 10: Teste de execução do cliente.

Na figura 11, temos um exemplo da informação multimédia recebida do servidor depois do cliente subscrever um canal.

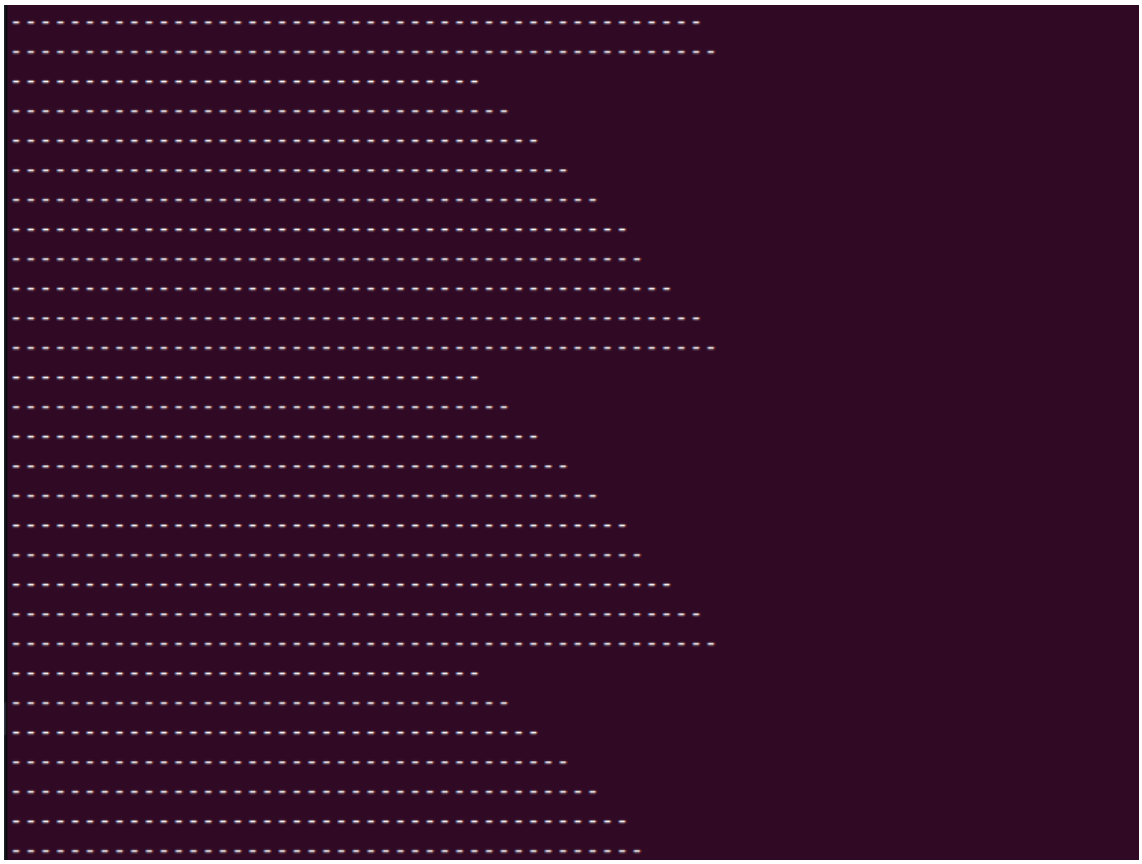


Figura 11: Teste de subscrição de em canal.

5 Conclusão

A concretização deste projeto foi bastante útil para aprofundar e aplicar os conhecimentos adquiridos nas aulas ao longo do semestre.
Numa primeira fase, obteve-se dificuldades na implementação das sockets do protocolo UDP, mas com alguma persistência foi possível superar as dificuldades deste trabalho.

Deste modo, é colossal o conhecimento que se levou após a realização deste projeto.

Referências

[1] BlackBoard.