FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Departamento de Engenharia Eletrotécnica

REDES INTEGRADAS DE TELECOMUNICAÇÕES I

2019 / 2020

Mestrado Integrado em Engenharia Eletrotécnica
e Computadores

4º ano
7º semestre

2º Trabalho Prático:
Troca de ficheiros por IPv4 e IPv6 com suporte *multicast*

http://tele1.dee.fct.unl.pt

Luis Bernardo

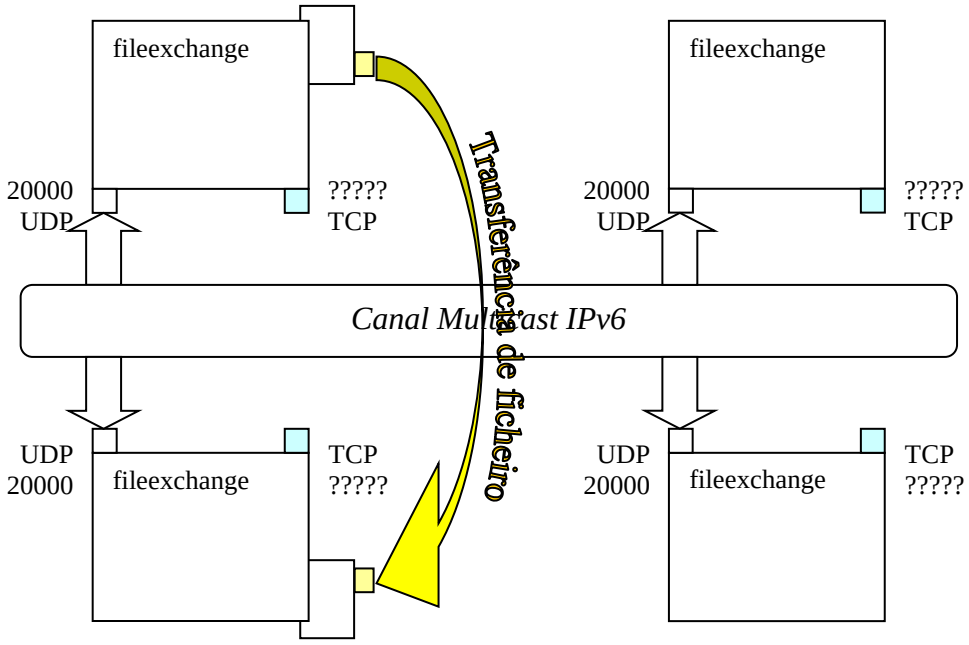
1. Objectivos

Familiarização com a programação de aplicações de pilha dupla (IPv6 e IPv4), o ambiente gráfico Gnome/Gtk+, e os mecanismos de gestão de tarefas e de sincronização entre tarefas no sistema operativo Linux. O trabalho consiste no desenvolvimento de uma aplicação de troca de ficheiros numa rede de pilha dupla (*dual stack*). A aplicação regista-se num endereço *multicast* IPv4 ou IPv6, enviando periodicamente um pacote de registo do nome do participante. Desta forma, cada participante tem uma imagem atualizada de todos os participantes na conversa, podendo comutar entre IPv4 e IPv6 livremente. A aplicação permite o envio de ficheiros entre dois participantes utilizando através de *sockets* TCP, usando tarefas (*threads*) POSIX para maximizar o paralelismo.

O trabalho consiste no desenvolvimento de um executável: ***fileexchange***.

2. Especificações

A aplicação *fileexchange* necessita de três parâmetros de configuração: o endereço IP Multicast do grupo (por omissão usa o endereço "ff18:10:33::1" para IPv6 e "225.0.0.1" para IPv4), o número de porto UDP (por omissão usa o porto 20000), e o nome do utilizador ("p" seguido do identificador de processo por omissão).



A aplicação *fileexchange* envia mensagens de registo de utilizador no grupo utilizando um *socket* datagrama. Usa um *socket* IPv6 para grupos IPv6 e um *socket* IPv4 para grupos IPv4. Para trocar ficheiros são usados um *socket* TCP para receber ligações (com um porto único) mais um número arbitrário de *sockets* TCP para enviar ficheiros, usados apenas dentro das tarefas.

A aplicação *fileexchange* começa por aguardar que o utilizador configure o canal (endereço IPv6 + número de porto) onde pretende escutar. Após o utilizador premir um botão, a aplicação arranca uma tarefa de registo periódico do nome do utilizador, e fica preparada para enviar ou receber ligações TCP.

Para poder enviar ficheiros, deve proceder inicialmente à escolha do ficheiro a enviar. Depois, pode selecionar um utilizador do grupo na interface gráfica, e iniciar a transferência do

ficheiro. Esta operação desencadeia o arranque de uma tarefa, que abre um novo *socket* TCP e trata do envio do conteúdo do ficheiro. Após a receção de uma nova ligação, também deve ser criada uma tarefa para receber o ficheiro. A janela principal deve listar todas as tarefas ativas, os ficheiros recebidos, e a percentagem de bytes transferidos. Um temporizador de inatividade permite lidar com a terminação abrupta do participante remoto da ligação. Na fase final do trabalho, pretende-se que os alunos configurem as propriedades do socket TCP de forma a reduzir ao mínimo o tempo de transferência de ficheiros.

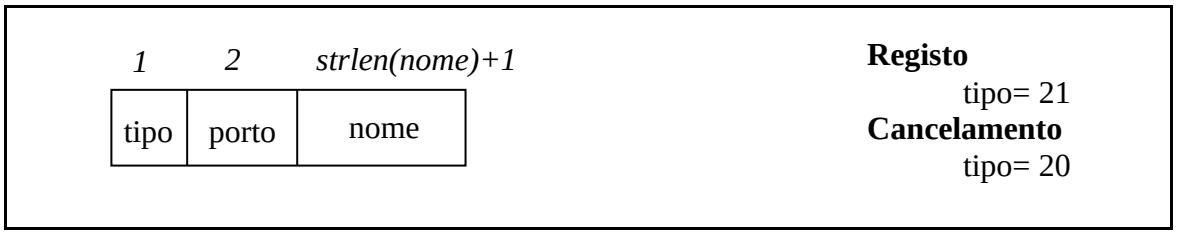
Para possibilitar o desenvolvimento do trabalho nas cinco semanas do trabalho, são fornecidos vários ficheiros, incluindo-se o ficheiro de especificação da interface gráfica "*gui_t2.glade*" com a janela principal do programa, mais um conjunto de módulos que realizam toda a leitura e escrita de dados da interface gráfica, e parte da lógica da aplicação.

2.1. Registo e cancelamento do nome

A partir do momento em que ficar ativa, a aplicação deverá enviar um pacote de registo do nome para o endereço *multicast* do domínio especificado (IPv4 ou IPv6), de 10 em 10 segundos. Este pacote identifica o utilizador, e simultaneamente, anuncia o endereço IPv4 ou IPv6 e o número de porto do socket TCP, usado nas transferências de ficheiros.

A mensagem enviada para o grupo consiste num octeto com o tipo de mensagem, seguida do número de porto do socket TCP, e de uma cadeia de carateres com o nome que se pretende registar (terminado com o carácter '\0'). A mensagem de cancelamento de registo tem uma estrutura semelhante.

```
Mensagem registo/cancelamento de nome: { sequência contígua de }
 unsigned char tipo;     // tipo de mensagem  -  Registo=21 /Cancelamento=20
 unsigned short porto;   // porto do socket TCP
 char *nome;             // array de carateres terminado por '\0'
```



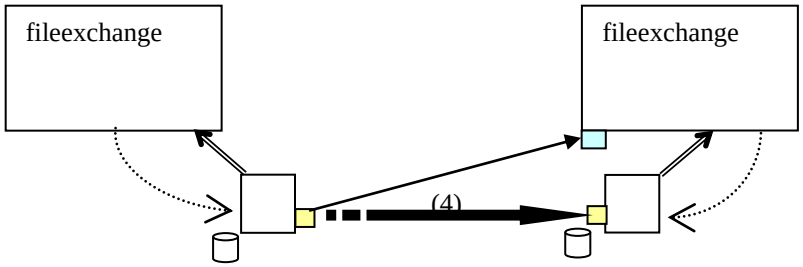
A aplicação *fileexchange* apresenta a lista de participantes do grupo numa tabela (componentes `GtkTreeView` e `GtkListStore`) permitindo uma visualização fácil do grupo.

Um participante no grupo pode terminar sem enviar o pacote de cancelamento de registo no nome. Caso não seja recebido nenhum pacote de registo do nome de um utilizador ao fim de pelo menos 20 segundos o utilizador deve ser retirado do grupo. Esta remoção pode ser realizada até 10 segundos depois de passarem os 20 segundos. Leia com atenção o conjunto de funções fornecidas, pois esta funcionalidade está parcialmente realizada.

2.2. Transferência de ficheiros

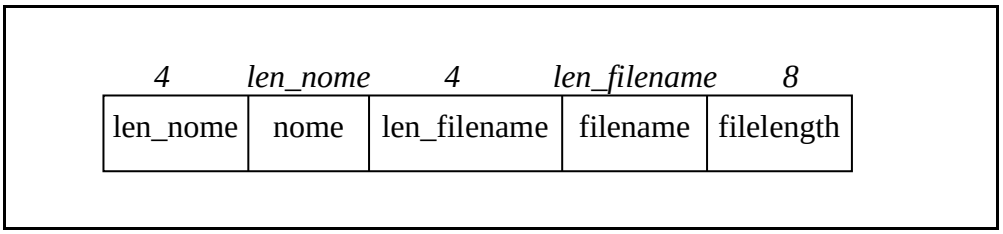
A aplicação deve permitir a troca de ficheiros entre dois participantes no grupo. A leitura e escrita dos ficheiros deve ser realizada em tarefas criados no emissor e no recetor do ficheiro. Após selecionar o ficheiro a enviar, o emissor deve criar uma tarefa de envio de ficheiro (1), que

cria um socket TCP temporário para enviar o ficheiro[1]. Após o cliente estabelecer ligação (2), é disparada no recetor a *callback* de aceitação de ligações. Esta *callback* deve também lançar uma tarefa (3), que fica em ciclo a receber o ficheiro e a escrevê-lo num ficheiro de saída.



No canal de comunicação entre o emissor e o recetor (4), o envio do ficheiro deve ser precedido do envio de um cabeçalho com a seguinte estrutura:

```
Cabeçalho de ficheiro: { sequência contígua de }
 int len_nome;          // Comprimento do nome do emissor (máx 128 bytes)
 char *nome;            // array de carateres terminado por '\0'
 int len_filename;      // Comprimento do nome do ficheiro (máx. 256 bytes)
 char *filename;        // array de carateres terminado por '\0'
 long long filelength;  // Comprimento do ficheiro a transmitir
```



As tarefas POSIX podem aceder à memória e à GUI da aplicação, correndo de forma concorrente. Mas o acesso concorrente de várias tarefas de receção e envio de ficheiro poderia levar a problemas de manipulação da interface gráfica e das listas partilhadas pelas várias tarefas (e.g. a lista de tarefas ativas). Desta forma, são usados semáforos (*Mutex*) para proteger todas as operações críticas.

3. Desenvolvimento da aplicação

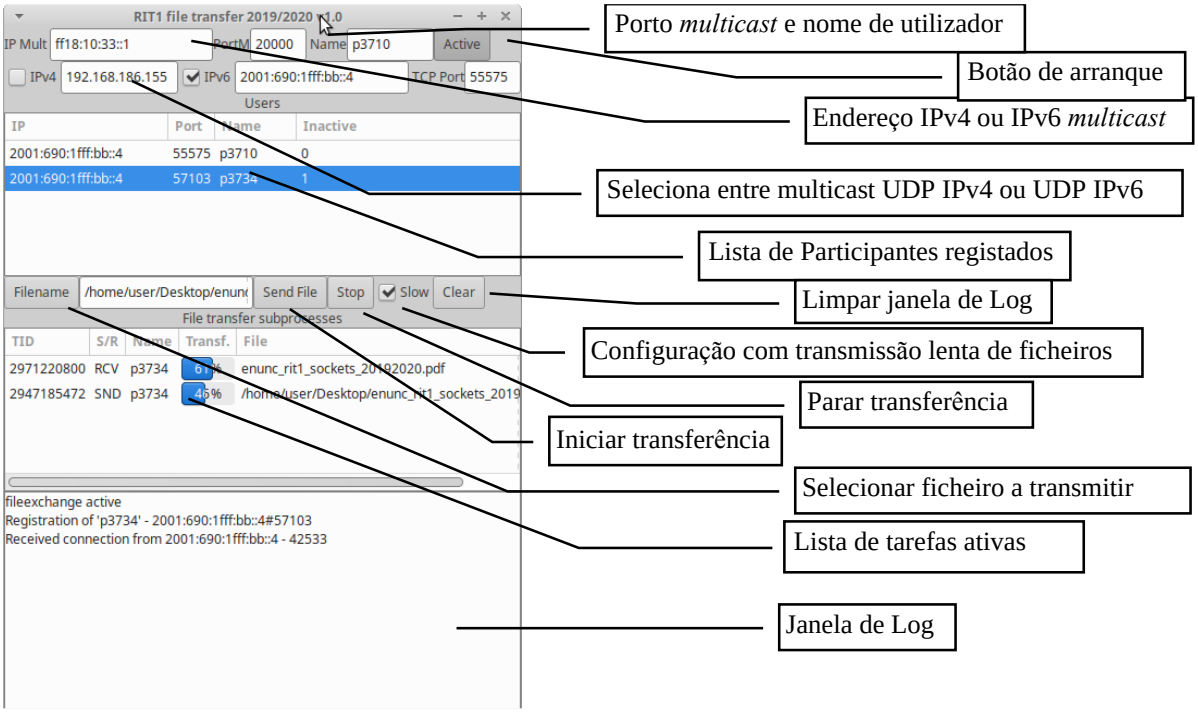
Para facilitar o desenvolvimento da aplicação é fornecido um programa de teste totalmente funcional (*demo_t2*), um ficheiro *glade* com a definição da interface gráfica do programa de teste representada abaixo, mais um conjunto de ficheiros descritos abaixo. Cada grupo pode fazer todas as modificações que quiser ao programa base. No entanto, recomenda-se que invistam o tempo na correta realização da aplicação proposta.

O código C fornecido está organizado em seis módulos:

- *sock.{c,h}* – Biblioteca de funções para lidar com *sockets* e com endereços IP (incluindo obter o endereço local) (completo);
- *file.{c,h}* – Biblioteca de funções para lidar com ficheiros (completo);
- *gui_g3.c, gui.h* – Biblioteca de funções para lidar com a lista de participantes, com a lista de transferências de ficheiros, registo de mensagens, e com a janela de seleção de ficheiros (completo);

[1] Note que pode usar um socket TCP IPv6 para se ligar a um socket TCP IPv4; não necessita de criar um socket TCP IPv4 adicional para esse fim.

- *thread.{c,h}* – Funções para lidar com threads. **Este módulo está incompleto** - só tem algumas funções de suporte;
- *callbacks.{c,h}* – Funções de arranque e paragem da aplicação, com iniciação dos *sockets*, *timers*, e mudança de modo entre IPv4 e IPv6. **Este módulo está incompleto** – falta completar parte das funções;
- *main.c* – Função de arranque da aplicação. Cria diretoria para guardar os ficheiros recebidos (completo).



O programa fornecido inclui todos os ficheiros completos exceto os ficheiros *thread.{c,h}* e *callbacks.{c,h}*, que devem ser completados pelos alunos.

No ficheiro *callbacks.{c,h}* está definido o tipo `Thread_Data` que é usado para guardar todos os dados de cada tarefa. Também está definida a variável `tcp_conn` (do tipo `GList *`, com uma lista de `Thread_Data`), com a lista de todas as tarefas ativas, e todas as funções que a manipulam.

Na comunicação *multicast* falta completar o arranque e paragem dos *sockets* IPv4, a configuração do relógio para envio periódico do nome, e a eliminação de participantes que deixam de enviar o pacote de registo do nome. Na comunicação de ficheiros está quase tudo por fazer.

A parte inicial do trabalho corresponde em ler e compreender o código fornecido, tornando realizável em pouco tempo a comunicação UDP IPv6. RECOMENDA-SE AOS ALUNOS QUE VEJAM O CÓDIGO FORNECIDO EM DETALHE, para conseguirem tirar total proveito dele.

O trabalho deve ser desenvolvido em várias fases distintas:

1. Completar a função `multicast_name` no ficheiro *callbacks.c*, de maneira a criar a mensagem de registo/cancelamento e enviá-la para a rede selecionada;
2. Programar o arranque e paragem do temporizador de envio do nome, que de 10 em 10 segundos envia o pacote de registo no socket UDP. Deve completar as funções `on_togglebuttonActive_toggled` e `close_all` no ficheiro *callbacks.c*;
3. Completar a função `remove_overdue` no ficheiro *callbacks.c*, de maneira a realizar a exclusão dos participantes que não enviam mensagens durante mais de 20 segundos;

4. Completar o arranque e paragem do socket UDP IPv4, usado para receber dados do grupo multicast IPv4. Deve completar as funções `close_sockUDP` e `init_socket_udp4` no ficheiro *callbacks.c*;
5. Programar a função `rcv_file_thread` no ficheiro *thread.c*, com o código para receber um ficheiro, guardando-o no disco local, atualizando o estado da ligação na GUI;
6. Completar a função `on_buttonSendFile_clicked` no ficheiro *callbacks.c*, de maneira a preparar os dados para lançar a tarefa de envio de ficheiro;
7. Completar a função `start_snd_file_thread` no ficheiro *thread.c*, de maneira a arrancar a tarefa de envio de ficheiro;
8. Programar a função `snd_file_thread` no ficheiro *thread.c*, com o código para ligar e enviar de um ficheiro, atualizando o estado da ligação na GUI;
9. Se tiver tempo, **otimize o débito na transferência de ficheiros**, modificando a dimensão dos buffers de envio e receção de dados no nível TCP, e os parâmetros dos sockets TCP[2]. No final, vai haver um **prémio de um valor em vinte na nota do trabalho para os DOIS trabalhos que tiverem menor tempo de transmissão de um ficheiro de teste**, medido durante a discussão final.

Para se conseguir chegar ao fim das cinco semanas do trabalho com tudo pronto é necessário utilizar todas as aulas práticas, devendo-se: concluir a fase (1) na primeira semana; ter iniciado a fase (4) na segunda semana; ter iniciado a fase (5) na terceira semana; ter iniciado a fase (8) na quarta semana; e concluir a fase (9, se tiver tempo) na última semana.

No mínimo, **pretende-se que todos os alunos cheguem ao fim da fase 5.** Procurem dividir tarefas pelos elementos do grupo de maneira a aproveitar melhor o tempo. As seguintes **fases podem ser feitas concorrentemente: (3) e (4); (5) e (6,7,8).**

Não se esqueça que no fim do semestre vai ser necessário entregar TODOS os trabalhos. Não deixe para a última semana o que pode fazer ao longo das primeiras semanas, porque **NÃO VAI CONSEGUIR FAZER TODO O TRABALHO NA ÚLTIMA SEMANA**.

Postura dos Alunos

Cada grupo deve ter em consideração o seguinte:

- Não perca tempo com a estética de entrada e saída de dados
- Programe de acordo com os princípios gerais de uma boa codificação (utilização de indentação, apresentação de comentários, uso de variáveis com nomes conformes às suas funções...) e
- Proceda de modo a que o trabalho a fazer fique equitativamente distribuído pelos membros do grupo.

[2] Observe-se que **o programa de teste fica intencionalmente lento quando tem a opção slow selecionada** porque inclui um '*usleep*' nos ciclos de envio/receção de ficheiro, que o faz "dormir" durante 50 ms entre envios de blocos do ficheiro, para além de não estar a configurar os *buffers* e parâmetros dos sockets TCP.