FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Departamento de Engenharia Electrotécnica

REDES INTEGRADAS DE TELECOMUNICAÇÕES I

2015 / 2016

Mestrado Integrado em Engenharia Electrotécnica
e Computadores

4º ano
7º semestre

2º Trabalho Prático:
Troca de ficheiros por *dual stack* com suporte *multicast*

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

Luis Bernardo

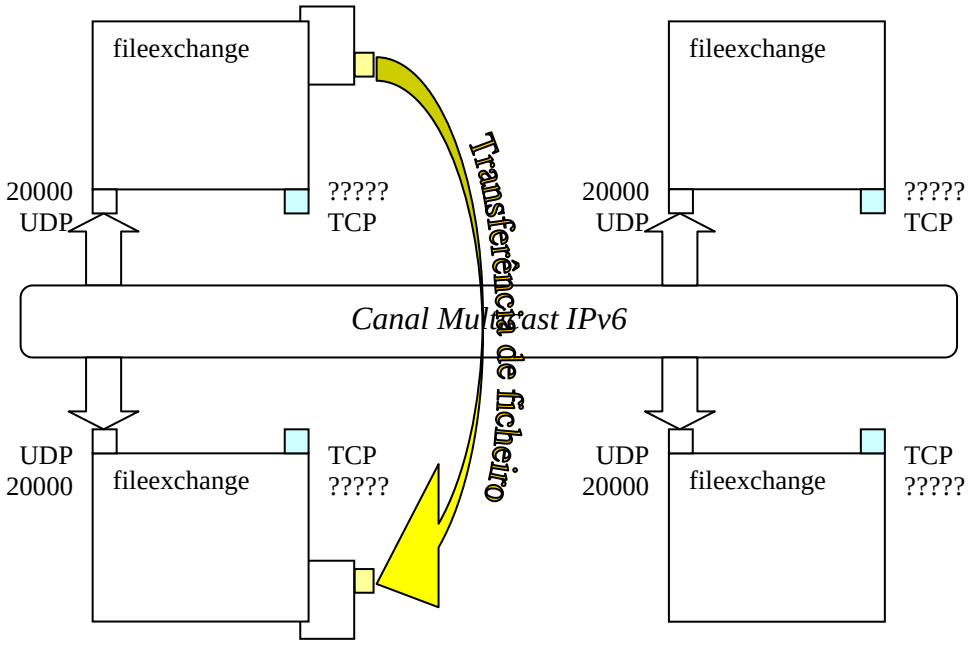
1. Objectivos

Familiarização com a programação de aplicações de pilha dupla (IPv6 e IPv4), o ambiente gráfico Gnome/Gtk+, e os mecanismos de gestão de processos e de comunicação entre processos 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 actualizada de todos os participantes na conversa, podendo comutar entre IPv4 e IPv6 livremente. A aplicação permite o envio de ficheiros entre dois participantes através de *sockets* TCP, lançando sub-processos 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 dos sub-processos.

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 lançamento de um sub-processo, que abre um novo *socket* TCP e trata do envio do conteúdo do ficheiro. Após a recepção de uma nova ligação, também deve ser criado um sub-processo para receber o ficheiro. A janela principal deve listar todos os sub-processos activos, 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 do ficheiro.

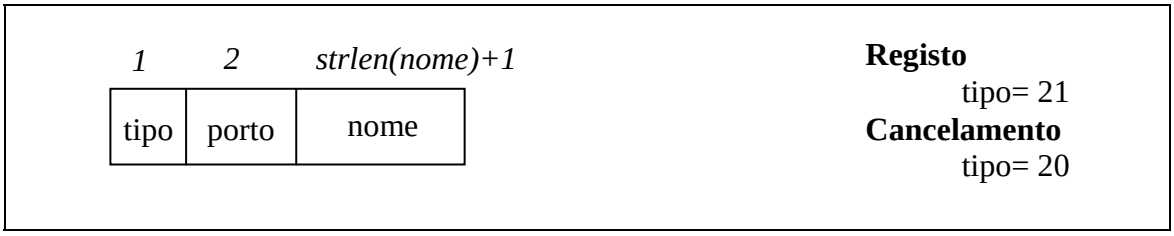
Para possibilitar o desenvolvimento do trabalho nas cinco semanas do trabalho, são fornecidos vários ficheiros, incluindo-se o ficheiro de especificação da interface gráfica "*gui_t2.glade*" com a janela principal do programa, mais um conjunto de módulos que realizam toda a leitura e escrita de dados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

2.1. Registo e cancelamento do nome

A partir do momento em que ficar activa, 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'
```



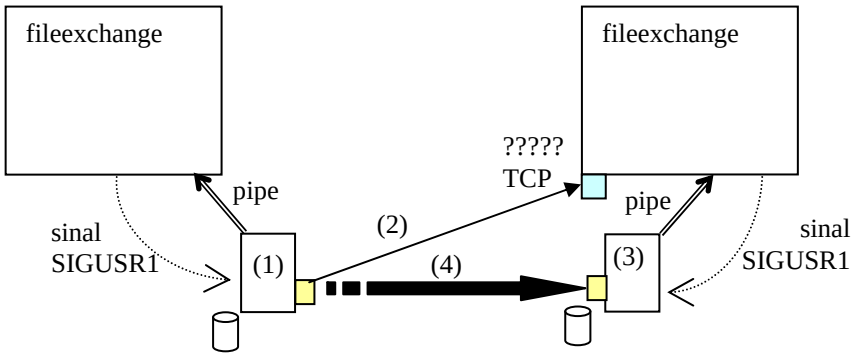
A aplicação *fileexchange* apresenta a lista de participantes do grupo numa tabela (componente `GtkCList`) permitindo uma visualização fácil do grupo.

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

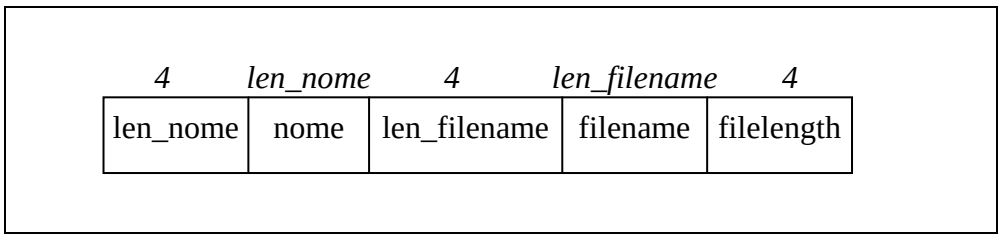
A aplicação deve permitir a troca de ficheiros entre dois participantes no grupo. A leitura e escrita dos ficheiros deve ser realizada em sub-processos criados no emissor e no receptor do ficheiro. Após seleccionar o ficheiro a enviar, o emissor deve criar um sub-processo de envio de

ficheiro (1), que cria um socket TCP temporário para enviar o ficheiro[1]. Após o cliente estabelecer ligação (2), é disparada no receptor a *callback* de aceitação de ligações. Esta *callback* deve também lançar um sub-processo (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 receptor (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 filelength;        // Comprimento do ficheiro a transmitir
```



Os sub-processos comunicam com os processos pai através de um *pipe* criado antes da separação do sub-processo, enviando mensagens com uma estrutura arbitrária, que informam o processo pai de: conteúdo do cabeçalho para preenchimento da interface gráfica (no caso do receptor); informação sobre evolução da transferência; fim da transferência (com informação sobre se houve erro, tempo de transferência e número de bytes). Como estas mensagens são internas ao programa, não são normalizadas no enunciado, devendo ser os alunos a definir a sua estrutura. O processo pai apenas vai comunicar com os sub-processos quando os quiser parar, utilizando o sinal SIGUSR1 para esse efeito.

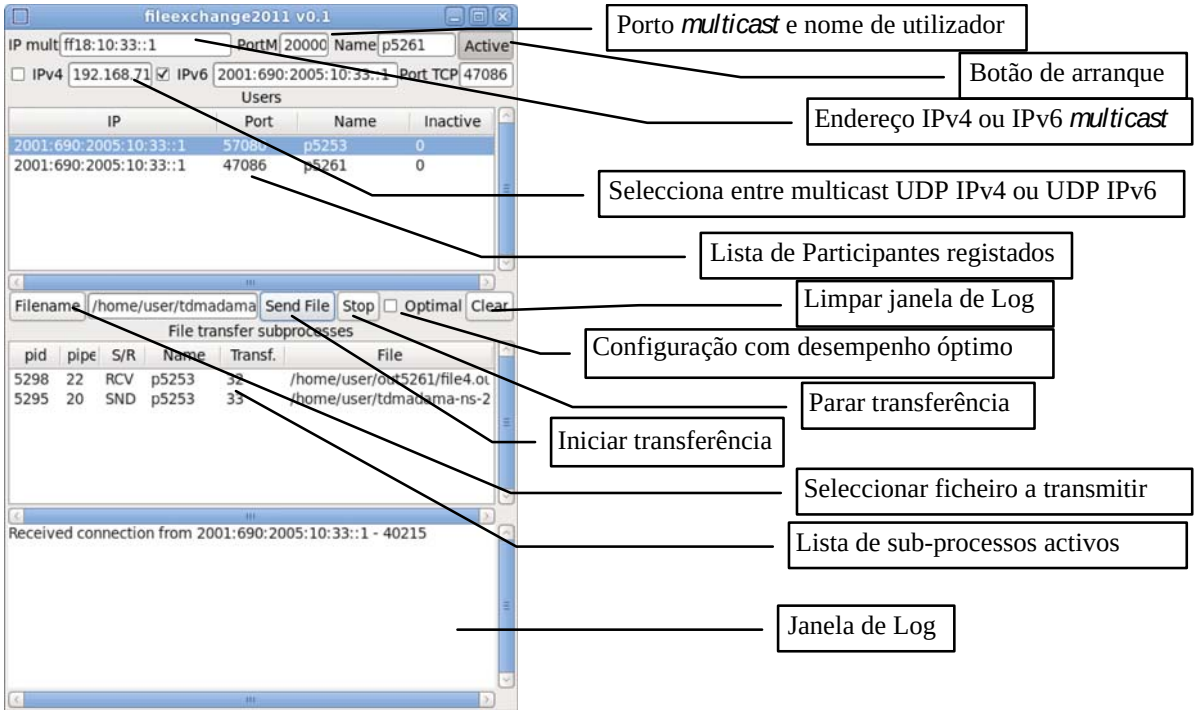
O processo pai também vai poder seguir o que está a ocorrer nos sub-processos através da rotina de tratamento do sinal SIGCHLD, corrida quando morre um dos seus filhos. Através da análise do motivo de saída (*exit* ou sinal) e do valor do *exit*, é possível detetar se o processo enviou uma mensagem através do *pipe*, ou se falhou.

3. Desenvolvimento da aplicação

Para facilitar o desenvolvimento da aplicação é fornecido um programa de teste totalmente funcional (*demo_t2*), um ficheiro *glade-2* com a definição da interface gráfica do programa de teste representada abaixo, mais um conjunto de ficheiros descritos abaixo. Cada grupo pode

[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.

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);
- *file.{c,h}* – Biblioteca de funções para lidar com ficheiros;
- *gui.{c,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 selecção de ficheiros;
- *subprocess.{c,h}* – Funções para lidar com sub-processos. **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 directoria para guardar os ficheiros recebidos.

O programa fornecido inclui todos os ficheiros completos excepto os ficheiros *subprocess.{c,h}* e *callbacks.{c,h}*, que devem ser completados pelos alunos. Na comunicação *multicast* falta completar o arranque e paragem dos *sockets*, 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. 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 iniciação e paragem dos *sockets* UDP e TCP, no botão de arranque;

2. Programação do temporizador de envio do nome, que de 10 em 10 segundos envia o pacote de registo no socket UDP. Envio de registo/cancelamento no botão de arranque;
3. Programação do tratamento de pacotes de registo/cancelamento na rotina de recepção de dados UDP, preenchendo a janela de participantes;
4. Programação do algoritmo de exclusão dos participantes que não enviam mensagens durante mais de 20 segundos;
5. Programação da rotina de envio de ficheiro, sem recorrer a sub-processos (de maneira a permitir corrigir os erros no debugger / envia 1 ficheiro de cada vez);
6. Programação das rotinas de lançamento do sub-processo de envio de ficheiro, incluindo as rotinas de tratamento do sinal SIGCHLD, SIGUSR1 (envia vários de cada vez);
7. Programação das rotinas de recepção do ficheiros. O ficheiro pode ser guardado num ficheiro de nome '*file000.out*', onde o número é incrementado por cada ficheiro recebido, na subdirectoria `out_dir`;
8. Programação das rotinas de lançamento do sub-processo de recepção do ficheiro, passando a poder ter recepção de ficheiros em paralelo com outras tarefas;
9. Definição de um protocolo para a comunicação entre os sub-processos e o processo pai. Programação da rotina de recepção de dados do *pipe*, no programa principal, e que recebe as mensagens dos sub-processos;
10. Se tiver tempo, teste de velocidade na transferência de ficheiro, modificando a dimensão dos buffers de envio e recepção de dados no nível TCP, e no ciclo de envio e recepção do ficheiro nos sub-processos. Observe-se que **o programa de teste fica intencionalmente lento quando não tem a optimização seleccionada** porque inclui um '*usleep*' no ciclo de envio de ficheiro, que o faz "dormir" durante 5 ms entre envios de blocos do ficheiro, para além de não estar a configurar os *buffers* dos sockets TCP.
11. Se tiver tempo, modifique o código dos botões de seleção entre IPv4 e IPv6, para reconfigurar o programa para mudar entre IPv4 e IPv6, mantendo todas as transmissões de ficheiro ativas. Numa fase inicial, isso será feito desligando e tornando a ligar a aplicação.

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; a fase (4) na segunda semana; ter iniciado a fase (6) na terceira semana; concluir a fase (8) na quarta semana; e concluir a fase (9, ou 11 se tiver tempo) 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 dois membros do grupo.