Projeto de Sistemas Operativos
2019-20
3º enunciado
LEIC-A/LEIC-T/LETI

O principal objetivo deste exercício é implementar uma aproximação da arquitetura final do TecnicoFS em modo utilizador, ilustrada na Figura 1. Esta solução permitirá que outros processos, externos ao processo servidor do TecnicoFS, possam invocar as operações do mesmo. Além disso, a funcionalidade do TecnicoFS é complementada com alguns aspectos que estavam omissos na solução construída até agora (nos 1º e 2º exercícios): o sistema de ficheiros passa a manter uma tabela de *i-nodes* e o conteúdo dos ficheiros, assim como a suportar um modelo de controlo de acesso aos ficheiros.

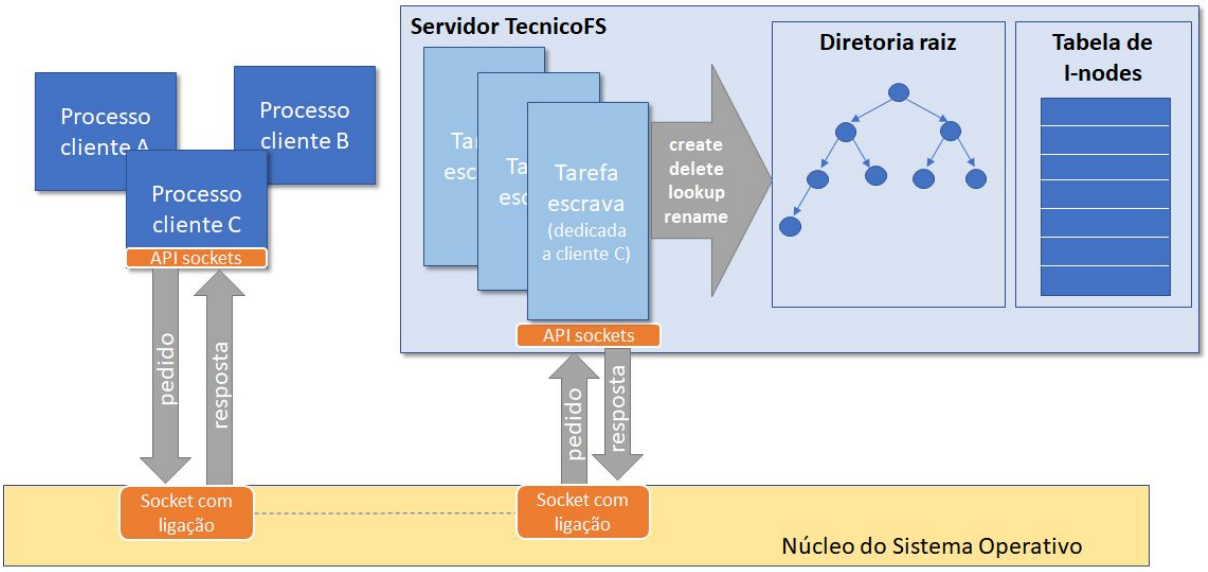

Figura 1: Arquitetura da solução do 3º exercício

Extensões ao servidor TecnicoFS

O servidor TecnicoFS deve ser estendido de diferentes formas, que são descritas de seguida.

1. *I-nodes* e conteúdo de ficheiros

O TecnicoFS passa a manter uma tabela de *i-nodes*. Esta tabela armazena os *i-nodes* dos ficheiros que existem na diretoria raíz, sendo indexada pelos *i-numbers* mantidos em cada nó da diretoria. Uma implementação completa da tabela de *i-nodes* é fornecida no site de SO.[1]

[1] Nota: Por simplicidade de código, a implementação fornecida para a tabela de i-nodes é assumidamente ineficiente em alguns aspetos, tais como: i) a sincronização recorre a um mutex único, o que limita o paralelismo; ii) a abordagem para determinar o próximo i-node livre é ineficiente; iii) o conteúdo dos ficheiros

Cada ficheiro passa a ter um dono, identificado pelo *user identifier* (UID) do processo cliente que criou o ficheiro (ver abaixo mais detalhes). Adicionalmente, a cada ficheiro estão associadas as permissões de acesso — leitura, escrita ou leitura/escrita — do dono do ficheiro e dos restantes utilizadores[2]. As permissões são definidas no momento em que o ficheiro é criado. Neste projeto não existem mecanismos para alterar as permissões. Todos estes elementos são guardados no *i-node* do ficheiro.

O *i-node* de um ficheiro inclui também um ponteiro para o conteúdo atual do ficheiro. Por simplificação, o conteúdo de um ficheiro está limitado a cadeias de caracteres textuais (*strings*), terminadas com '\0'.

2. Comunicação com processos cliente

O servidor TecnicoFS deixa de carregar comandos a partir de ficheiro. Em vez disso, passa a ter um socket *Unix* do tipo *stream*, através do qual recebe e aceita ligações solicitadas por outros processos, que designamos de processos cliente.

O executável do TecnicoFS passa a receber os seguintes argumentos de linha de comandos:

```
tecnicofs nomesocket outputfile numbuckets
```

O novo argumento, *nomesocket*, indica o nome que deve ser associado ao *socket* pelo qual o servidor receberá pedidos de ligação. Os restantes argumentos têm o mesmo significado que nos enunciados anteriores.

A tarefa inicial é responsável por inicializar o *socket* e aceitar pedidos de ligação que cheguem através do mesmo. Sempre que há uma nova ligação estabelecida com um processo cliente, é criada uma nova tarefa escrava[3] que fica encarregue de executar e responder aos pedidos enviados por esse cliente. O período em que a ligação entre o processo cliente e a tarefa escrava está ativa, será designado por "sessão". Antes de receber pedidos, a tarefa escrava cria uma tabela de ficheiros abertos que será usada durante a sessão. A tabela de ficheiros abertos é um vetor com 5 entradas. Fica ao critério de cada grupo definir qual o conteúdo de cada entrada da tabela de ficheiros abertos. Assim que a ligação seja fechada, a tarefa escrava liberta a tabela de ficheiros abertos e termina.

3. Terminação do servidor

O processo servidor deve passar a terminar ordeiramente quando recebe o *signal* SIGINT (causado por Ctrl+C). Mais precisamente, a terminação ordeira deve ser implementada da seguinte forma:

- Quando o processo do servidor TecnicoFS recebe um *signal* SIGINT, este deve ser tratado pela tarefa inicial (i.e., a tarefa que está a aceitar novos pedidos de ligação).
- A partir do momento em que chegue um *signal* desse tipo, a tarefa inicial deve deixar de aceitar pedidos de ligação.
- No entanto, como pode haver sessões ativas, o servidor TecnicoFS só deve terminar quando todas as sessões ativas terminarem. Nesse momento, o procedimento habitual de terminação deve ser executado (impressão do tempo acumulado de execução e impressão do conteúdo atual da diretoria para o ficheiro de saída).

não pode ser modificado parcialmente nem estendido, apenas totalmente substituído. Está fora do âmbito do 3º projeto corrigir estas ineficiências, pelo que tal não será valorizado na avaliação.

[2]Por simplicidade, em relação ao Unix, não se está a considerar as permissões do grupo.

[3]Consequentemente, deixa de existir uma pool fixa de tarefas escravas criada no início da execução.

Notas:

- Para forçar que o *signal* SIGINT seja tratado pela tarefa inicial (em vez de uma tarefa escrava), pode recorrer à função *pthread_sigmask*.
- Por simplicidade, pode assumir que a solução correrá em sistemas que oferecem a semântica BSD no que respeita ao tratamento dos signals.

API cliente do TecnicoFS

Para permitir que outros processos possam interagir com o TecnicoFS, existe uma interface de programação (API), em C, a qual designamos por *API cliente* do TecnicoFS. Esta API permite ao cliente ter programas que estabelecem uma sessão com um servidor TecnicoFS e, durante essa sessão, invocar funções para aceder e modificar o sistema de ficheiros. O ficheiro .h com as assinaturas das funções da API, assim como os códigos de erro por elas retornados, é fornecido no site de SO. Esse ficheiro não deve ser alterado pelos alunos.

As funções da API cliente do TecnicoFS são descritas de seguida. Começando pelas funções para estabelecer e terminar sessão com o servidor TecnicoFS:

- `int mount(char *address)`
  Estabelece uma sessão com o servidor TecnicoFS localizado no endereço passado como argumento. O endereço corresponde a um *socket* do domínio Unix, ou seja, um nome de ficheiro (existente no sistema de ficheiros nativo da máquina onde corre o cliente).
  Devolve erro se o cliente já tiver uma sessão ativa com algum servidor TecnicoFS.
  Retorna 0 em caso de sucesso ou um código de erro caso contrário.
- `int unmount()`
  Termina uma sessão ativa. Devolve erro caso não exista sessão ativa.
  Retorna 0 em caso de sucesso ou um código de erro caso contrário.

As funções seguintes só podem ser chamadas quando o cliente tiver uma sessão ativa. Todas elas devolvem erro caso sejam chamadas quando o cliente não tem sessão ativa ou quando a comunicação com o servidor falhe (por exemplo, caso o servidor tenha terminado entretanto).

- `int create(char *filename, int ownerPermissions, int othersPermissions)`
  Cria um novo ficheiro com as permissões indicadas, cujo dono é o UID efetivo do processo cliente. As permissões são definidas da seguinte forma: 0 - nenhuma, 1 - escrita apenas, 2 - leitura apenas ou 3 - leitura/escrita.
  Devolve erro caso já exista um ficheiro com o mesmo nome.
  Retorna 0 em caso de sucesso ou um código de erro caso contrário.

- `int delete(char *filename)`
  Apaga o ficheiro indicado em argumento.
  Devolve erro caso o UID efetivo do processo cliente não seja o mesmo do dono do ficheiro ou caso o ficheiro não exista.
  Retorna 0 em caso de sucesso ou um código de erro caso contrário.

- `int rename(char *filenameOld, char *filenameNew)`
  Renomeia o ficheiro com o nome indicado no 1º argumento para o novo nome no 2º argumento.
  Devolve erro caso não exista um ficheiro com o nome antigo passado por argumento, o UID do dono do ficheiro antigo não seja o UID efetivo do processo cliente ou o novo nome já

esteja atribuído a outro ficheiro.
Retorna 0 em caso de sucesso ou um código de erro caso contrário.

- `int open (char *filename, int mode)`
  Abre o ficheiro passado por argumento no modo indicado (1 - escrita, 2 - leitura, 3 - leitura/escrita).
  Devolve erro caso o ficheiro não exista, o UID efetivo do processo cliente não tenha permissão para aceder ao ficheiro no modo indicado ou o processo cliente já tenha esgotado o número máximo de ficheiros que pode abrir.
  Retorna o descritor do ficheiro aberto (inteiro não negativo) em caso de sucesso ou um código de erro caso contrário.
- `int close(int fd)`
  Fecha o ficheiro previamente aberto (por este mesmo processo cliente) com o descritor de ficheiro passado por argumento.
  Devolve erro caso o descritor não se refira a nenhum ficheiro aberto.
  Retorna 0 em caso de sucesso ou um código de erro caso contrário.

- `int read (int fd, char *buffer, int len)`
  Copia o conteúdo do ficheiro aberto *fd* para *buffer,* cuja dimensão máxima é indicada pelo argumento *len.* A *string* colocada em *buffer* é terminada em '\0', ou seja, no máximo são copiados *len - 1* caracteres do ficheiro para o *buffer*.
  Devolve erro caso o ficheiro não esteja aberto ou não tenha sido aberto em modo de leitura ou leitura/escrita.
  Retorna o número de caracteres lidos (excluindo o '\0') em caso de sucesso ou um código de erro caso contrário.

- `int write (int fd, char *buffer, int len)`
  Substitui o conteúdo do ficheiro aberto por uma cópia da *string* contida no *buffer* passado como argumento, até ao máximo de *len* bytes ou até ser encontrado um '\0' no *buffer* (o que ocorrer primeiro).
  Devolve erro caso o ficheiro não esteja aberto ou não tenha sido aberto em modo de escrita ou leitura/escrita.
  Retorna 0 em caso de sucesso ou um código de erro caso contrário.

Nota: a enumeração dos casos de erro de cada função não é exaustiva. A solução pode também devolver outros erros que se justifiquem.

Protocolo de pedido-resposta

As mensagens de pedidos, enviadas por cada processo cliente, devem consistir em *strings* (logo, terminadas em '\0'). Os vários campos das mensagens de pedido são separados por um espaço (' '). Para cada mensagem, o processo recetor deve ler do *socket stream* os bytes suficientes até alcançar o terminador '\0'.

A solução deve assumir que o processo cliente só envia um pedido após ter recebido resposta para o pedido anterior. Ou seja, o caso em que existem dois pedidos simultaneamente pendentes para leitura no mesmo *socket stream* está fora da especificação do sistema, logo não tem de ser suportado na solução.

A sintaxe de cada mensagem de pedido deve seguir a seguinte especificação:

Função da API cliente	Mensagem de pedido
`create`	"`c filename permissions`" (onde permissions consiste na concatenação de ownerPermissions e othersPermissions ex: "c abc 32")
`delete`	"`d filename`"
`rename`	"`r filenameOld filenameNew`"
`open`	"`o filename mode`"
`close`	"`x fd`"
`read`	"`l fd len`"
`write`	"`w fd dataInBuffer`"

Com a exceção da função "read", a mensagem de resposta aos pedidos acima consiste num inteiro (do tipo *int,* enviado em binário pelo canal) contendo o código de retorno de cada função. No caso da função "read", o código de retorno deve ser seguido dos dados lidos do ficheiro (ou seja, um *int* seguido de uma *string* terminada em '\0').

Experimente

No site de SO encontra um exemplo de um programa cliente que usa a API cliente do TecnicoFS. Este exemplo executa uma sequência de chamadas ao TecnicoFS e verifica se os retornos estão de acordo com o esperado.

Compile esse exemplo e faça a "*linkagem*" com a sua implementação da API cliente. Depois, execute-o sobre o seu servidor do TecnicoFS e confirme se os resultados observados pelo cliente exemplo são os esperados.

Componha os seus próprios exemplos de programas cliente para testar outras funcionalidades.

Planeamento, entrega e avaliação

Não sendo obrigatório, sugere-se que a resolução siga as seguintes fases:

1. Começar por implementar as funções *mount* e *unmount* (tanto do lado do cliente como do lado do servidor, incluindo a criação de tarefa escrava dedicada a cada nova sessão);
2. Implementar as funções *create*, *rename* e *delete* (incluindo a integração da BST com a tabela de *i-nodes* e o controlo de acesso por UID).
3. Implementar as funções open, close, read e write.
4. Implementar a terminação ordeira do servidor.

Os alunos devem submeter um ficheiro no formato *zip* com o código fonte. Devem existir duas sub-diretorias, chamadas `client` e `server`, cada uma contendo o código relativo ao cliente e servidor, respetivamente.

A diretoria `client` deve conter apenas a implementação da API cliente (ficheiros .h e .c respetivos). A diretoria `server` deve conter o código fonte do servidor TecnicoFS e o respetivo ficheiro *Makefile*. Por simplificação, a avaliação deste exercício só terá em conta a variante

`tecnicofs-rwlock`. Assim sendo, a *Makefile* deve ser alterada para que o comando make (sem argumentos) produza apenas este executável. Como antes, o comando *make clean* deve limpar todos os ficheiros resultantes da compilação do projeto.

Tal como nas entregas anteriores, o arquivo *zip* submetido não deve incluir outros ficheiros tais como binários.

O exercício deve **obrigatoriamente** compilar e executar nos computadores dos laboratórios. O uso de outros ambientes para o desenvolvimento/teste do projeto (e.g., macOS, Windows/WSL) é permitido mas o corpo docente não dará apoio técnico a dúvidas relacionadas especificamente com esses ambientes.

A submissão é feita através do Fénix até ao dia ~~29/novembro~~ **2/dezembro** às 23h59.

A avaliação será feita de acordo com o método de avaliação descrito no site da cadeira.

Os alunos devem consultar regularmente a plataforma piazza. É lá que serão prestados esclarecimentos sobre este enunciado.