Instituto Superior Técnico

Departamento de Engenharia Informática

Enunciado do Projecto de Sistemas Operativos

LEIC/LERC

2007/2008

Biblioteca Multitarefa em Modo Utilizador.

Sistema de Ficheiros Distribuído *Simple NFS*.

Índice

Introdução

Este trabalho tem por objectivo familiarizar os alunos com os seguintes aspectos do desenvolvimento de sistemas operativos:

- Gestão e sincronização entre tarefas
- Comunicação entre processos
- Arquitectura do sistema de ficheiros

O projecto consiste na definição da arquitectura e programação de uma versão simplificada de um sistema de ficheiros distribuído inspirada na arquitectura do sistema de ficheiros distribuído da SUN NFS – Network File System. O sistema *Simple NFS* permite que vários processos (designados por clientes) partilhem ficheiros que são armazenados por um processo servidor.
O projecto será desenvolvido de forma modular, com várias etapas faseadas no tempo. Nomeadamente, os alunos deverão realizar os seguintes módulos:

- Estender uma biblioteca de gestão de tarefas utilizador – *sthreads* – de modo a suportar diversas funcionalidades frequentes em sistemas de tarefas e necessárias para a realização do resto do projecto.
- Analisar e programar o processo servidor de ficheiros.
- Analisar e programar o suporte para a comunicação entre os processos cliente e o processo servidor.
- Definir e programar uma biblioteca de funções (*API – Application Programming Interface*) através da qual as aplicações acedem ao sistema de ficheiros e cuja interface esconde (tanto quanto possível) das aplicações o facto de o sistema ser distribuído.

Os alunos não terão que desenvolver todos os módulos de raiz; nalguns casos, pretende-se que estendam o código que será fornecido pelo corpo docente. Os alunos devem desenvolver o projecto de acordo com o faseamento proposto neste enunciado e que se relaciona com as etapas de avaliação.

O trabalho está estruturado em quatro partes, que devem ser realizadas de forma sequencial.

Parte I – Biblioteca *sthreads*

Objectivos

A biblioteca *sthreads* é uma biblioteca que permite gerir tarefas que se executam em modo utilizador (corotinas), no contexto de um único processo gerido pelo sistema operativo. É fornecida aos alunos uma versão desta biblioteca que suporta a criação de tarefas e o seu escalonamento, usando uma política de *round-robin*. Pretende-se estender o código fornecido com as seguintes funcionalidades:

- Escalonamento baseado em *time-slices*.
- Primitivas de sincronização entre tarefas (semáforos).
- Primitivas adicionais de gestão de tarefas.

Concretização

O conjunto de funcionalidades a adicionar é apresentado em três grupos. No Anexo A encontram-se informações complementares para a concretização de cada um destes grupos de funcionalidades.

1. Escalonamento com base em time-slices. Pretende-se acrescentar o mecanismo de escalonamento baseado em *time-slices* às tarefas, fazendo evoluir o escalonamento inicial que apenas implementa uma política de *round-robin*. A biblioteca disponibilizada suporta apenas comutação provocada explicitamente através da invocação da rotina `sthread_yield()`. Para concretizar esta nova politica de escalonamento é necessário acrescentar código que se executa de cada vez que ocorre um *signal*. O material fornecido para esta parte do trabalho ajuda nos seguintes aspectos:

- Como gerar e tratar *signals*;
- Primitivas para ligar e desligar a rotina de tratamento do *signal* de forma a garantir exclusão mútua necessária para a correcta codificação das funções de gestão de tarefas e funções de sincronização.

2. Mecanismos de sincronização. Acrescentar os mecanismos de sincronização semáforos à biblioteca utilizador. O esqueleto das funções encontra-se no ficheiro *sthread_user.c*. Os programadores que usem a biblioteca *sthreads* podem simular trincos utilizando semáforos inicializados com uma unidade. Nesta parte do trabalho tem que:

- Implementar as estruturas para os semáforos (`struct _sthread_sem`);
- Implementar as operações dos semáforos (`sthread_user_sem_init()`, `sthread_user_sem_wait()`,`sthread_user_sem_post()`,`sthread_user_sem_free()`).

3. Gestão das tarefas. Pretende-se acrescentar as seguintes funções de gestão de tarefas:

- A função de espera pela terminação de outra tarefa – *(join)* (`sthread_user_join()`);
- A função adormecer tarefa (`sthread_user_sleep(int time)`);

Para realizar esta parte do trabalho deve usar o seguinte material do pacote *simplethreads*:

- Estruturas de dados que representam o estado das tarefas;
- O esqueleto destas rotinas providenciadas no ficheiro *sthread_user.c*.

Parte II – Cliente SNFS

Objectivos

Pretende-se desenvolver a biblioteca SNFS que permite a uma aplicação cliente aceder ao sistema de ficheiros de um servidor remoto SNFS. O código das aplicações cliente deve ser ligado com a biblioteca SNFS que está dividida em duas camadas (ver Anexo A-3):

- Uma camada de suporte à comunicação com o servidor (API SNFS).
- Uma camada de interface com a aplicação (Interface Sistema de Ficheiros).

Para permitir o teste destas funcionalidades será fornecida uma concretização funcional (mas não completa) do servidor. O servidor será lançado da linha de comandos com dois argumentos: o primeiro é o caminho para o *socket* Unix do servidor, o segundo é opcional e indica o caminho do ficheiro com uma imagem com que inicializar o sistema de ficheiros do servidor.

Concretização

1. API SNFS. Esta camada consiste num conjunto de rotinas que i) formata as mensagens SNFS e ii) comunica com o servidor através de *sockets* domínio Unix sem ligação (SOCK_DGRAM). A utilização deste tipo de sockets possui a vantagem de i) delimitar automaticamente as mensagens e de ii) enviar os dados nas mensagens sem necessidade de conversão.

Para esta parte é disponibilizado o material seguinte:

- Servidor SNFS funcional.
- Formato das mensagens do protocolo SNFS para todos os serviços.
- Esqueleto da biblioteca SNFS para os clientes.

2. Interface Sistema de Ficheiros. Esta camada deverá oferecer aos programadores uma interface de programação semelhante àquela que é disponibilizada pela biblioteca standard da linguagem C. Por sua vez, deverá utilizar a API SNFS sempre que necessitar de comunicar com o servidor. Internamente, esta camada deve gerir os ficheiros abertos pela aplicação cliente.

Para simplificar o projecto não é necessário considerar que o sistema trata estes dois aspectos que são imprescindíveis em situações reais.

- Acesso concorrente à biblioteca por várias tarefas do mesmo processo cliente.
- Acesso ao mesmo ficheiro por dois processos cliente independetes.

Parte III – Servidor multi-tarefa

Objectivos

O servidor SNFS que é fornecido aos alunos funciona com uma única tarefa. Os alunos devem alterar o servidor para funcionar em modo multi-tarefa. A arquitectura multi-tarefa do servidor SNFS deve obedecer ao seguinte desenho:

- Existe um número limitado de tarefas consumidoras (T_C) processam e responder aos pedidos dos clientes.
- Existe uma única tarefa (T_P) que descobe quando existem novos pedidos e que os distribui pelas tarefas cliente.

Concretização

A tarefa T_P tem de se sincronizar com as tarefas T_C.

- Quando chega um novo pedido, T_P necessita de assegurar que uma tarefa T_C, e apenas uma, fica responsável por este pedido.
- Se todas as tarefas T_C estiverem ocupadas, o mecanismo de sincronização deve assegurar-se que o pedido não é esquecido.

O número de tarefas T_C deverá ser definido em tempo de compilação.

Parte IV – Funcionalidades do Sistema de Ficheiros

Objectivos

O servidor de SNFS fornecido deve ser estendido com mais a funcionalidade, o sistema de ficheiros do servidor SNFS fornecido aos alunos é muito simples e possui as seguintes limitações:

- O tamanho máximo dos ficheiros é de 10 blocos.
- Suporta apenas um único directório raíz onde está armazenada todos os meta-dados que descrevem os ficheiros.

As alterações a fazer devem eliminar as limitações da concretização fornecida.

Concretização

O sistema de ficheiros fornecido possui as seguintes características:

- Tamanho dos blocos tem dimensão fixa de 512 bytes. O número de blocos é definido em tempo de compilação (um valor razoável para testes é de 64 Kblocos que ocupa 32 MB em memória).
- A arquitectura do sistema de ficheiros é baseada numa simplificação dos i-nodes Unix , com 10 entradas directas para blocos de dados. O tamanho de um i-node é de 64 bytes. O número de i-nodes por omissão é de 64 (a tabela de i-nodes ocupa, por isso 8 blocos), mas este valor pode ser definido em tempo de compilação.
- Os directórios são ficheiros estruturados sob a forma de uma tabela de entradas. Cada entrada tem 16 bytes e é constituída por: nome do ficheiro/directório e número do i-node. Os nomes são limitados a 14 caracteres e o número do i-node a 2 bytes. Para simplificar a listagem dos directórios remotamente, considera-se que os directórios podem ocupar, no máximo 4 blocos, isto é suportam até 128 entradas.
- A disposição dos dados nos blocos é a seguinte:
    - Bloco 0: reservado para o *bitmap* de blocos livres.
    - Bloco 1: reservado para o *bitmap* de i-nodes livres.
    - Blocos 2-9: reservados para a tabela de i-nodes.
    - Blocos = 10: para os dados dos ficheiros e directórios.

Os alunos deverão concretizar as seguintes extensões ao sistema de ficheiros:

- Estender os i-nodes com um nível de indirecção de forma a aumentar a dimensão máxima permitida para os ficheiros.
- Permitir estruturação hierárquica de directórios.

A realização desta parte do trabalho implica alterar as estruturas de dados e a implementação dos serviços do servidor SNFS disponibilizado. Novamente no intuito de simplificar o projecto não é necessário desenvolver as funções de libertação de blocos ou entradas nos directórios pois não se considera a função de Eliminar ficheiros/directórios.

Entrega e Avaliação

A entrega do trabalho será feita em três fases: duas entregas intercalares e a entrega final.

Entregas intercalares

As entregas intercalares serão verificadas nas aulas práticas correspondentes de cada grupo:

- **1º Checkpoint**: entrega a 22 de Outubro, até às 12h, via Fénix, visualização na aula prática seguinte – Parte I do trabalho.
- **2º Checkpoint**: entrega a 19 de Novembro, até às 12h, via Fénix, visualização na aula prática seguinte – Partes I, II e III do trabalho.

As entregas intercalares podem contar para subir a nota de acordo com a seguinte fórmula:

> Nota final do projecto = MAX(Nota original do projecto; 0.8*Nota original do projecto + 0.2*Nota das entregas intercalares)

onde "Nota original do projecto" decorre da avaliação normal do projecto, e a nota das entregas intercalares será uma avaliação generosa do esforço dispendido para completar o trabalho a tempo das entregas intercalares (por exemplo, na nota das entregas intercalares seremos bastante benevolentes em relação à qualidade do desenho e da implementação do sistema).

Entrega Final

A entrega final consiste na entrega das Partes I, II, III e IV e implica obrigatoriamente duas fases:

1 – Via electrónica

A entrega por via electrónica será efectuada no dia 7 de Dezembro de 2007 até às 12h, através do sítio da cadeira. O formato do ficheiro de entrega com o código será indicado oportunamente.

2 – Envelope

Para além da entrega electrónica deve ser entregue até às 15h do mesmo dia um envelope fechado identificado com o dia, hora, turno e número de grupo:

O código impresso em 2 páginas por folha, frente e verso, sem linhas cortadas e identado.

Essa entrega é realizada nos seguintes locais:

- Pólo Alameda: Reprografia do DEI;
- Pólo Tagus Park: Gabinete de Apoio às Licenciaturas.

Visualização e Discussão do Projecto

Os projectos serão visualizados durante a semana de 10 a 16 de Dezembro nas aulas práticas.

Após a entrega dos projectos será afixado no sítio WWW da cadeira um horário de discussões dos projectos. As discussões dos projectos realizam-se na semana de 17 a 21 de Dezembro. De notar que nas discussões serão feitas perguntas sobre a implementação das rotinas que são fornecidas à partida, portanto não basta usá-las, há que entendê-las.

A informação sobre a entrega e avaliação aqui descrita está sujeita a alterações. Se tal suceder, serão afixados avisos na página da disciplina. Por este motivo recomenda-se a sua consulta periódica.

Anexo A – Pacote *sthreads+snfs*

1. Material fornecido

É distribuído o pacote *sthreads+snfs-chk1-skel.tgz* com o esqueleto base, e encontra-se na página da cadeira. O material dado consiste num pacote que permite criar tarefas que correm em modo utilizador e que é necessário para desenvolver o projecto até ao 1º Checkpoint. Outro material necessário será distribuído e adicionado ao directório base em momento oportuno. Este pacote é composto pelo seguinte:

Directoria / Ficheiro	**Conteúdo**
sthread_lib/	Biblioteca de tarefas
sthread_lib/sthread_user.c	Onde vão ser implementadas as funções necessárias para a biblioteca de tarefas.
sthread_lib /sthread_ctx.{c,h}	Módulo para criar novas pilhas de execução e para comutar entre elas
sthread_lib /sthread_switch_i386.h	Funções assembly para comutar entre pilhas e para salvaguardar registos
sthread_lib/sthread_time_slice.{c,h}	Suporte para gerar signals e para controlá-los
sthread_lib/queue.{c,h}	Módulo de filas (*queues*) para gestão de tarefas
include/	Contém sthread.h, config.h que são as interfaces públicas, a incluir em programas que usem ou comuniquem com os programas ou bibliotecas referidas.
test-sthreads/	Contém vários testes para a biblioteca sthreads

As rotinas no ficheiro sthread_ctx.h realizam toda a manipulação da pilha, alterações ao PC (Program Counter), guardam registos e outras manipulações de baixo nível. O objectivo da Partes I do trabalho é a administração dos contextos de execução das tarefas, pelo que apenas é necessário implementar as funções que se encontram no sthread_user. Durante a implementação modificar apenas o ficheiro sthread_user.c. Não modificar sthread_ctx_t directamente, usando, em vez disso, as rotinas declaradas em sthread_ctx.h. Vendo o sistema em camadas (Figura 1) apenas tem que implementar o "rectângulo a cinzento".

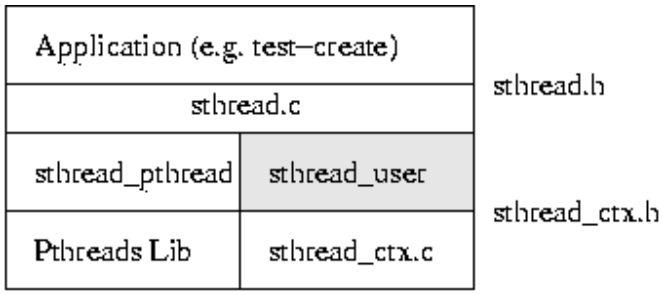


Figura 1

Na camada superior encontra-se a aplicação que vai usar o pacote sthread (através da API definida em sthread.h). sthread.c vai então chamar as rotinas implementadas neste trabalho em sthread_user.c ou as rotinas em sthread_pthread.c que fazem uso das pthreads (alterando a variável PTHREADS nas Makefiles). O sthread_user.c por sua vez é construído em cima das rotinas do sthread_ctx (como descrito em sthread_ctx.h).

As aplicações (camada superior) não devem usar mais rotina nenhuma da biblioteca excepto as definidas em sthread.h. As aplicações não podem usar rotinas definidas noutros ficheiros nem podem "saber" como estão implementadas as tarefas. As aplicações apenas pedem para criar tarefas e podem requerer yield() ou exit. Também não devem manter listas de tarefas a correr. Isso é a função do "rectângulo cinzento".

De igual forma, o rectângulo cinzento – sthread_user.c – não deve saber como sthread_ctx está implementado. Deve usar as rotinas definidas em sthread_ctx.h.

De seguida apresentam-se algumas notas sobre as várias partes da implementação da biblioteca de tarefas-utilizador.

2. Notas sobre as Funções da Biblioteca Sthreads

- `sthread_create` cria a tarefa que se irá executar quando for seleccionada pelo despacho. Qualquer tarefa só deve deixar de correr quando ela própria executar o `sthread_yield()`;
- Use as rotinas fornecidas em sthread_ctx.h. Não necessita escrever nenhum código assembly, nem manipular registos, nem entender detalhadamente como está organizada a pilha;
- A rotina `sthread_new_ctx()` cria as estruturas de dados necessárias para reprentação do contecto de uma nova tarefa. Não recebe nenhum argumento (ao contrário da rotina `sthread_user_create()`). Por isso não é possível criar uma nova pilha directamente com a rotina do utilizador. É necessário criar uma rotina que não recebe argumentos mas de alguma forma invoca a rotina do utilizador com os argumentos do utilizador;

- A rotina que é passada para `sthread_create()` corresponde ao programa principal da tarefa pelo que se esta terminar, tem que se assegurar que os recursos são libertados (ou seja, `sthread_exit()` tem que ser chamado quer explicitamente pela rotina que é passada para `sthread_create()` quer implicitamente após a rotina terminar);
- Deve libertar todos os recursos quando a tarefa termina. Não deve no entanto libertar a pilha de uma tarefa que se encontre ainda a correr (nota: para libertar a pilha use `sthread_free_ctx()`);
- A tarefa inicial deve ser tratada como qualquer outra tarefa. Deve por isso ser criada um estrutura `sthread_t` para ela para que exista algum lugar onde colocar o estado da tarefa em caso de se querer pará-la.;
- Tenha cuidado com o uso de variáveis locais após chamar `sthread_switch()` já que os seus valores podem ser diferentes de anteriormente (é uma pilha de execução diferente);
- Apesar do uso de variáveis globais não ser recomendado, nalguns casos terão mesmo de ser usadas.

Nos semáforos:

- Compreenda como bloquear uma tarefa, fazendo-a esperar numa fila. Como obtém a tarefa que se bloqueou? Como altera o contexto dela para passar para outra tarefa? Como volta a corrê-la?
- Quando se desbloqueia uma tarefa, não ocorre imediatamente uma mudança de tarefa. A tarefa fica executável e será executada quando seleccionada pelo despacho;
- Todos os testes fornecidos no pacote, excepto os de time-slices, devem funcionar ao acabar esta parte.

Para inicializar o sistema de preempção no suporte de time-slices, tem que começar por chamar a rotina `sthread_time_slices_init()`, a qual tem dois argumentos: uma função que corre a cada signal, e o período em microsegundos. A rotina `sthread_time_slices_init` tem que ser chamada no fim da rotina `sthread_user_init()`. Faça com que o escalonador de tarefas transite para uma tarefa diferente de cada vez que ocorre um *signal*.

Para colocar sincronização no sistema de tarefas pense no que acontece se ocorrerem *signals* em vários pontos do seu código. Por exemplo, não quer que ocorra um *signal* no meio da execução de `yield()` ou de uma mudança de tarefas. A melhor forma de isso nunca acontecer é desactivar o tratamento do *signal*. Para isso usar a rotina `splx(int splval)`, onde `splx(HIGH)` desactiva a rotina de tratamento do *signal* e `splx(LOW)` activa-a.

Existem na biblioteca duas outras primitivas de sincronização: `atomic_test_and_set` e `atomic_clear` que apenas seriam essenciais num ambiente de multiprocessador com tarefas núcleo, e neste caso podem ser ignoradas.

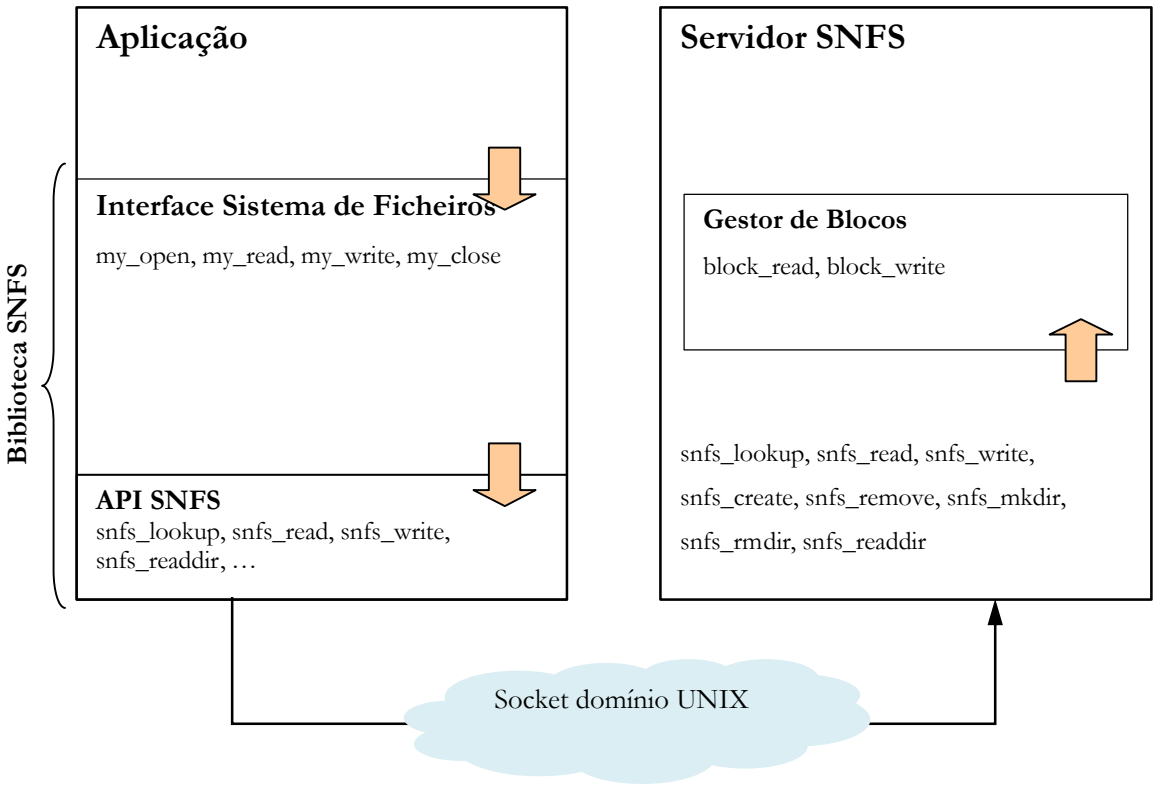
O temporizador foi implementado de forma aos *signals* apenas serem tratados se ocorrerem no código do utilizador (aplicação ou biblioteca de tarefas). *Signals* que ocorram dentro de printfs ou outras chamadas de sistema são ignorados. *Signals* no código assembly dentro do `sthread_switch` são também ignorados.

Outras Notas

- Comece por iniciar a preempção para correr uma função que apenas imprime algo no ecrã. Isto permite verificar que os *signals* estão a funcionar como esperado;
- Se desactivar os *signals* não se esqueça de as activar em todos o caminhos possíveis do seu programa. Provavelmente vai querer desactivar os *signals* durante todo o tempo que estiver dentro de um yield e activá-las ao completar o `sthread_switch`. De notar que `sthread_switch` pode retornar para dois locais diferentes: para a linha a seguir a uma chamada a um `sthread_switch` ou para a sua função de começo da tarefa quando comuta para uma nova tarefa pela primeira vez. Active os *signals* nos dois locais;
- Não deve executar código da aplicação com os *signals* desligados;
- Para ajudar no *debugging* carregue em CTRL-\ em qualquer altura para ver o número de *signals* gerados.
- Verifique que o teste para *time-slices* funciona (test-time-slices.c) e verifique que todos os outros testes funcionam ainda.
- Uns bons valores para o período das interrupções são entre 10 e 50 microsegundos.
- Para comparar programas com e sem preempção comente a chamada à rotina `sthread_time_slices_init()`.

3. Arquitectura do Cliente-Servidor SNFS

O sistema de ficheiros será implementado de acordo com a seguinte arquitectura.



Notas:

- O servidor SNFS não mantém estado entre os pedidos. Logo, cada pedido é acompanhado de toda a informação necessária para que possa ser atendido.

4. Protocolo SNFS

O protocolo SNFS define a comunicação entre clientes e servidores SNFS. O SNFS deve suportar o subconjunto dos serviços do protocolo NFS Versão 2 descritos na Tabela 1. A lista completa de serviços do protocolo NFS Versão 2 encontra-se do RFC 1094 (http://tools.ietf.org/html/rfc1094).

Serviço	Argumentos	Resultado	Descrição
SNFS_LOOKUP	fhandle dir filename name	stat status se STAT_OK fhandle file unsigned fsize	Obtém o identificador "fhandle" e respectivo tamanho "fsize" do ficheiro ou directório "name" localizado no directório "dir", se a resposta obtida for STAT_OK. Se "fhandle" for 0, refere-se ao directório raiz.
SNFS_READ	fhandle file unsigned offset unsigned count	stat status: se STAT_OK byte data	Devolve até "count" bytes de "data" do ficheiro "file", a partir do byte "offset" a contar do início do ficheiro. O primeiro byte do ficheiro corresponde ao offset 0.
SNFS_WRITE	fhandle file unsigned offset unsigned count byte data	stat status: se STAT_OK unsigned fsize	Escreve "data" a partir do byte "offset" a partir do início do ficheiro "file". O primeiro byte do ficheiro está no offset 0. A operação de escrita é atómica. Os dados de uma escrita não serão misturados com os dados da escrita de outro cliente. Se for bem sucedida, a escrita devolve a dimensão actual do ficheiro em "fsize"
SNFS_CREATE	fhandle dir filename name	stat status se STAT_OK fhandle file	Cria o ficheiro "name" no directório dado por "dir". Os atributos iniciais do ficheiro são dados por "attributes". Se o resultado é STAT_OK, o ficheiro foi criado com sucesso e "file" e "attributes" contém o "fhandle" e os atributos do ficheiro. Caso contrário a operação falhou e nenhum ficheiro foi criado.
SNFS_MKDIR	fhandle dir filename name	stat status se STAT_OK fhandle file	Cria o novo directório "name" no directório "dir". A resposta STAT_OK indica que o directório foi criado com sucesso e "file" contém o fhandle do directório. Caso contrário, a operação falhou e o directório não foi criado.
SNFS_READDIR	fhandle dir unsigned count	stat status se STAT_OK entry* list unsigned count	Retorna um número variável de entradas do directório até "count" bytes do directório dado por "dir". Se o valor retornado é STAT_OK, então segue-se de um número variável de "entry"s. Cada entry é uma estrutura com o nome da entrada (ficheiro ou directório), tamanho do nome e indicação se o nome se trata de ficheiro/directório.

Tabela 1: Serviços do protocolo SNFS

5. API do Cliente

A interface de programação do sistema de ficheiros SNFS é semelhante à oferecida pela biblioteca standard da linguagem C:

- `int my_init_lib();`

  Inicializa as estruturas internas da biblioteca de ficheiros.

- `int my_open(char * nome, int flags);`

  Devolve um inteiro que identifica o ficheiro em operações posteriores (handle). O Argumento *flags* é usado para modificar o comportamento da função, sendo que apenas existe a *flag* com valor 1, que indica que o ficheiro deve ser criado caso já não exista.

- `int my_read(int fileID, char * buffer, unsigned numBytes);`

  Lê a partir de um ficheiro, para um *buffer* em memória, um número especificado de *bytes*. Devolve o número de *bytes* lidos, ou zero caso esteja no final do ficheiro.

- `int my_write(int fileID, char * buffer, unsigned numBytes);`

  Escreve para um ficheiro, o conteúdo de um *buffer* em memória, com o tamanho especificado.

- `void my_close(int fileID);`

  Fecha o ficheiro identificado pelo argumento *fileID*.

- `int my_listdir(char* path, char **filenames, int* numFiles);`

  Escreve para o ponteiro **filenames* o endereço de memória onde estão contidos os nomes dos ficheiros que se encontram no directório *path* do sistema de ficheiros, separados por '\0', e no inteiro numberOfFiles a quantidade de dos mesmos ficheiros.

  O espaço de memória onde são escritos os nomes deve ser alocado pela função my_listdir e libertado pela função que a chama, assim que não precisar mais dos nomes.

Da mesma forma que as funções mencionadas anteriormente, estas funções devem devolver um código de erro negativo caso encontrem situações anómalas.