Trabalho de Projeto

1ª Fase

1 Introdução

A componente teórico-prática da disciplina de Sistemas Operativos pretende familiarizar os alunos com alguns dos problemas envolvidos na utilização dos recursos de um Sistema Operativo. O projeto de avaliação será realizado utilizando principalmente a linguagem de programação C, as APIs de Linux e POSIX (*Portable Operating System Interface*) e a programação do `makefile`.

O objetivo geral do projeto será o desenvolvimento de uma aplicação em C, chamada ***SOchain***, para simular um sistema de transações de um *token* cripto denominado ***SOT*** (*SO Token*). Esta aplicação permitirá que um utilizador, através da `Main`, controle um conjunto de carteiras (*wallets*) e realize transações de SOT entre elas. O sistema envolverá múltiplos processos cooperativos, representando três entidades principais: (i) a `Main`, que interage com o utilizador e permite a este solicitar transações em nome das carteiras; (ii) as `Wallets`, que assinam apenas as transações cuja carteira de origem sejam elas próprias; e (iii) os `Servers`, que validam, processam e registam as transações. O fluxo de chamadas entre os processos envolve: (i) o utilizador pedir à `Main` a criação de uma transação, a qual será encaminhada para a respetiva `Wallet` de origem, (ii) a assinatura da transação pela respetiva `Wallet` de origem e a sua submissão para processamento pelos `Servers`, e (iii) a validação, execução e armazenamento da transação no *ledger*[1] pelos `Servers`. O sistema inclui um *buffer* de resposta, permitindo que a `Main` recupere os comprovativos das transações mediante os pedidos do utilizador.

O *SOchain* será realizado em duas fases. A primeira fase tem como objetivo fundamental a familiarização com a linguagem C, a criação do ficheiro `makefile` (para a compilação do projeto), a gestão de processos e a alocação de memória. Nesta fase inicial, os alunos poderão trabalhar sem sincronização ou com apenas uma instância por entidade, focando-se nas estruturas de dados e na preparação do sistema. Os servidores poderão imprimir as informações das transações no `stdout`, em vez de escrever diretamente no arquivo do *ledger*. Neste primeiro enunciado, é feita uma apresentação geral do *SOchain*, juntamente com a informação específica de suporte à realização da primeira fase. Na segunda fase, será disponibilizado um novo enunciado que complementará a informação contida neste documento, introduzindo a sincronização entre os processos, o tratamento de sinais, o uso de temporizadores e as operações de escrita e leitura sobre os ficheiros.

2 Funcionamento Geral

Como referido acima, no modelo do sistema do *SOchain*, existem três tipos de participantes: `Main`, `Wallets` e `Servers`. Cada execução do *SOchain* representa um ciclo de transações, onde pode haver uma ou mais instâncias de `Wallets` e `Servers`, conforme configurado pelo utilizador do sistema. Para além destas duas entidades, existe também o processo principal (`Main`), que gere a interação com o utilizador, permitindo a criação e a consulta de transações. Estas entidades e as relações entre elas estão representadas na Figura 1.

A `Main` oferece um menu com opções para que o utilizador possa interagir com o *SOchain* e gerir as transações entre as carteiras. O menu contém sete opções:

1. **`bal id`** – Obtém o saldo atual (*balance*) da carteira (*wallet*) cujo identificador é `id`.
2. **`trx src_id dest_id amount`** – O utilizador solicita a criação de uma transação em que a carteira de origem (`src_id`) envia uma determinada quantidade (`amount`) de SOT *tokens* para a carteira de destino (`dest_id`). Como resultado, obtém o identificador da transação criada (`id`).
3. **`rcp id`** – Obtém o comprovativo (*receipt*) da execução de uma transação específica, identificada por `id`. O resultado retorna a instância da transação com todas as suas propriedades preenchidas.
4. **`stat`** – Apresenta o estado atual das variáveis do `info_container` (descrito no final desta secção).
5. **`help`** – Mostra as informações de ajuda sobre as operações disponíveis.
6. **`end`** – Termina a execução do sistema *SOchain*.

[1] Um *ledger* é um livro caixa onde são registadas todas as transações realizadas nas *blockchains*. No caso da *SOchain*, o *ledger* nada mais é do que um ficheiro em modo texto com os registos das transações em formato de log (uma transação por linha).

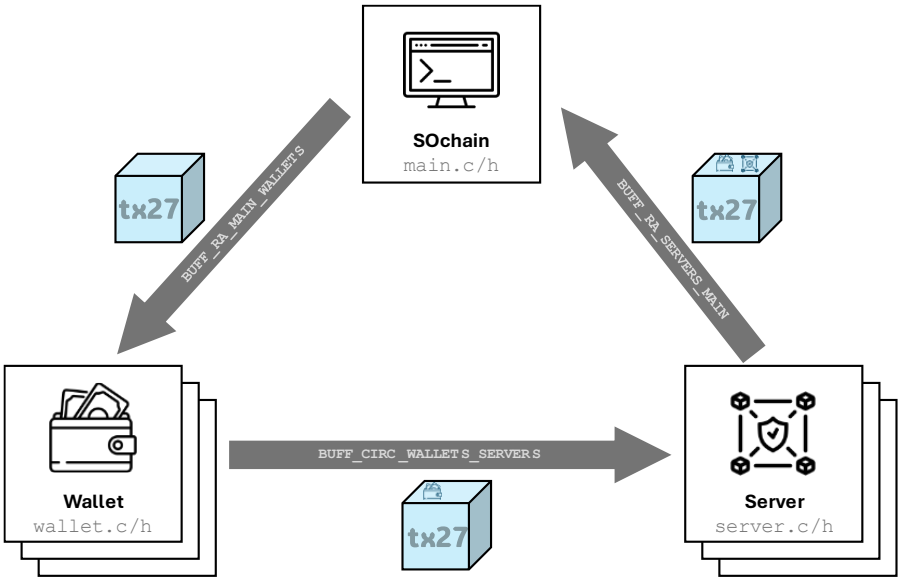

Figura 1: Visão geral do sistema SOchain e as interações entre os participantes.

O utilizador pode pedir (na interface de linha de comandos do *SOchain*) com o comando **`bal`**, a qualquer momento, para verificar o saldo de qualquer carteira do sistema cujo identificador seja o `id` passado como argumento da operação. Neste caso, a `Main` acederá à estrutura de dados partilhada que conterá os saldos de cada carteira e retornará o valor que lá estiver atualmente.

O utilizador pode pedir a criação de uma transação através da operação **`trx`**, especificando os identificadores das carteiras de origem e destino envolvidas (`src_id` e `dest_id`) e a quantia (`amount`) de SOT a ser transferido entre elas. A `Main` recebe este pedido de operação do utilizador e cria uma nova instância da estrutura de dados (`struct`) que representa uma transação e preenche os campos que já tem informação disponível. A `Main` possui um contador que guarda quantas outras transações já haviam sido criadas, o que lhe permite definir qual o identificador da transação que está a ser criada nesta operação (e.g., a Transação 27 na Figura 1). Esta transação criada é então escrita pela `Main` num *buffer* de comunicação de acesso aleatório (identificado por `BUFF_RA_MAIN_WALLETS`), onde ela ficará disponível para a leitura por parte das entidades que representam as carteiras (`Wallets`). Apesar de todas as `Wallets` terem acesso para lerem deste buffer, somente a `Wallet` cujo `id` corresponde ao `src_id` da transação é que pode ler a transação criada na operação.

Após a leitura, a carteira de origem assina a transação (colocando o seu `id` no campo da assinatura da `Wallet`) para confirmar que reconhece e aprova a transação criada pela `Main` na operação solicitada pelo utilizador. De seguida, esta `Wallet` escreve a transação assinada por ela num buffer de comunicação do tipo circular (identificado por `BUFF_CIRC_WALLETS_SERVERS`), onde ela ficará disponível para a leitura por parte de qualquer entidade que representa os servidores (`Servers`). Neste caso, qualquer servidor poderá ler a transação e processá-la.

Quando um servidor lê do *buffer* a transação assinada pela `Wallet` de origem, ele primeiro verifica a validade da mesma (e.g., se as carteiras envolvidas na transação existem e se há saldo suficiente na carteira de origem). Caso a transação seja válida, o servidor (i) efetua a transferência dos *tokens* SOT da carteira de origem para a carteira de destino, (ii) assina a transação com o seu id, (iii) regista a transação no ledger e (iv) escreve a transação executada e assinada por ele para um buffer de comunicação de acesso aleatório (identificado por `BUFF_RA_SERVERS_MAIN`), onde ela ficará disponível para leitura da `Main` mediante um pedido de obtenção do comprovativo (*receipt*) por parte do utilizador (ver o comando `rcp` a seguir). Caso a transação seja inválida, ela é descartada pelo servidor e não constará nos registos do *ledger* da *SOchain*.

O utilizador pode pedir com o comando **`rcp`** o comprovativo da execução de uma determinada transação. Este comprovativo nada mais é do que a instância da transação assinada pela carteira de origem e pelo servidor que a executou, a qual estará disponível no buffer de comunicação de acesso aleatório `BUFF_RA_SERVERS_MAIN` caso a transação tenha sido solicitada e executada corretamente e o comprovativo ainda não tenha sido lido nenhuma vez. A `Main` lê do *buffer* esta instância da transação com todas as assinaturas e imprime todas as informações contidas nela (i.e., o id da transação, os ids das carteiras de origem e destino, a quantia, os ids da carteira que assinou a transação e do servidor que a processou) como forma de entregar ao utilizador o comprovativo da mesma. Cada comprovativo pode ser lido no máximo uma vez, pois quando a Main lê do buffer, a transação é retirada do buffer e não voltará a ser inserida lá por nenhum servidor pois ela não será novamente executada.

Das restantes operações, a operação **stat** imprime o estado atual das variáveis do `info_container`, permitindo inspecionar todo o sistema *SOchain*. A operação **help** apresenta o menu com as operações disponíveis ao utilizador. A operação **end** imprime as estatísticas finais (i.e., o mesmo que o `stat`) e termina a execução do sistema *SOchain*.

O `info_container` é uma estrutura de dados utilizada para centralizar uma série de informações sobre o sistema *SOchain* para facilitar a sua gestão e monitorização. Ele deve guardar as seguintes informações:

- Os argumentos passados ao programa *SOchain*. Estes basicamente incluem o saldo inicial de cada carteira, o número de carteiras no sistema, o número de servidores, o tamanho dos buffers de comunicação e o número máximo de transações que o sistema poderá processar.
- O PID de cada processo `Wallet` e `Server` criado no sistema.
- O número total de transações criadas no sistema *SOchain* até o momento.
- O número de transações assinadas por cada `Wallet` até o momento.
- O número de transações validadas e processadas por cada `Server` até o momento.
- O saldo atual de cada `Wallet`.

3 Descrição Específica

A Fase 1 do projeto consiste na concretização (i.e., programação) dos seguintes módulos:

- **Main (`main.c/h`)** – Módulo principal que gere os outros módulos e realiza a interação com o utilizador, permitindo solicitar transações entre as carteiras, obter os comprovativos das transações, visualizar o estado do sistema e terminar a execução do *SOchain*.
- **Gestão de Processos (`process.c/h`)** – Contém funções para criar e destruir processos, representando as diferentes entidades do sistema (`Wallets` e `Servers`).
- **Gestão de Memória (`memory.c/h`)** – Inclui funções para a criação de memória dinâmica e partilhada, bem como operações sobre os *buffers* circulares e de acesso aleatório, fundamentais para a comunicação entre as entidades.
- **Wallet (`wallet.c/h`)** – Responsável por assinar as transações cuja carteira de origem (`src_id`) seja a própria `Wallet` e enviá-las para o buffer que será lido pelos `Servers` para o processamento das transações.
- **Server (`server.c/h`)** – Responsável por validar, processar e armazenar transações, garantindo a atualização dos saldos das carteiras e registando as transações no *ledger*.

Para cada um destes módulos, é fornecido pelos professores um ficheiro `.h` com os cabeçalhos das funções, **que não podem ser alterados**. As concretizações das funções definidas nos ficheiros `X.h` (onde `X` representa o nome do ficheiro) terão de ser desenvolvidas pelos grupos nos respetivos ficheiros `X.c`, utilizando os algoritmos e métodos que considerarem adequados. Se necessário, ou caso seja solicitado, os alunos poderão criar um ficheiro de cabeçalho `X-private.h` para incluir definições adicionais, cujas implementações deverão ser incorporadas também nos respetivos `X.c`.

Juntamente com o enunciado da 1ª Fase do projeto, será disponibilizado aos alunos um ficheiro ZIP contendo a estrutura de diretórios do projeto e todos os cabeçalhos necessários para a implementação dos módulos acima mencionados. Para auxiliar no início do desenvolvimento do projeto, é também fornecida uma função `main` na Figura 2, a qual os alunos poderão utilizar como base e alterar caso achem necessário.

4 Estrutura do Projeto e makefile

O projeto deve ser organizado segundo a seguinte estrutura:

```
SOCHAIN/
      bin/
      inc/
      obj/
      src/
      makefile
```

O diretório `bin` deverá conter o executável `SOchain` gerado pela execução do comando `make`. O ficheiro `makefile` necessário para a execução do comando `make` será desenvolvido pelos alunos e deve ser colocado na raiz do diretório `SOCHAIN`. O diretório `inc` deverá conter os ficheiros `.h` com a definição das estruturas de dados e declarações de funções. Estes ficheiros não podem ser alterados (à exceção dos ficheiros `-X-private.h`). O diretório `obj` deverá conter os ficheiros objeto gerados (ficheiros `.o`) pela execução do `makefile`. Por fim, o diretório `src` deverá conter os ficheiros fonte (ficheiros `.c`) com os códigos desenvolvidos pelos alunos.

```
int main(int argc, char *argv[]) {
	//init data structures
	struct info_container* info = allocate_dynamic_memory(sizeof(struct info_container));
	struct buffers* buffs = allocate_dynamic_memory(sizeof(struct buffers));

	//execute main code
	main_args(argc, argv, info);
	create_dynamic_memory_structs(info, buffs);
	create_shared_memory_structs(info, buffs);
	create_processes(info, buffs);
	user_interaction(info, buffs);

	//release memory before terminating
	destroy_shared_memory_structs(info, buffs);
	destroy_dynamic_memory_structs(info, buffs);
	return 0;
}
```

Figura 2: Exemplo de código para a função main do SOchain.

5 Desenvolvimento e Testes

Para efeitos de teste, é fornecido aos alunos o executável **`SOchain_profs`**, desenvolvido pelos professores da disciplina e compilado numa máquina virtual contendo a distribuição Linux instalada nos laboratórios de aula da FCUL. É esperado que, tendo sido introduzidos os mesmos argumentos e operações, tanto o executável desenvolvido pelos alunos como o desenvolvido pelos professores retornem resultados idênticos ou muito semelhantes.
Exemplo de utilização do executável **`SOchain`** (e do **`SOchain_profs`**):

`$ ./SOchain init_balance n_wallets n_servers buff_size max_txs`

Onde:

- **`init_balance`** define a quantidade inicial de SOT atribuída a cada carteira (`Wallet`) no início da execução do programa.
- **`n_wallets`** é o número total de carteiras no sistema.
- **`n_servers`** é o número de servidores responsáveis pelo processamento das transações.
- **`buff_size`** é o tamanho máximo dos *buffers* (i.e., o número máximo de transações que podem estar ao mesmo tempo nos buffers de comunicação entre os processos).
- **`max_txs`** é o número máximo de transações que poderão ser executadas em todo o sistema.

Recomenda-se aos alunos que comecem os testes com um número reduzido de transações e processos (e.g., 1 carteira, 1 servidor e 10 transações no máximo,). Caso o código funcione corretamente com este exemplo, devem aumentar progressivamente a escala, adicionando mais carteiras, servidores e transações.

De notar que nesta 1ª fase do projeto, e no caso de haver múltiplos servidores, pode acontecer que mais de um servidor tente validar e processar a mesma transação ao mesmo tempo. Este problema demonstra a necessidade de sincronização entre os processos, o que será resolvido na 2ª fase do projeto.

Após o arranque da aplicação, o utilizador deve interagir com o sistema *SOchain* por meio de um menu com as operações `bal`, `trx`, `rcp`, `stat` e `end`, que foram explicadas na Secção 2. Deve também ser possível, por meio da opção `help`, imprimir as informações sobre as operações disponíveis.

6 Entrega

A entrega da primeira fase do projeto tem de ser feita de acordo com as seguintes regras:

1. Colocar todos os ficheiros do projeto, de acordo com a estrutura apresentada na Secção 0, bem como um ficheiro `README` onde os alunos podem incluir informações que julguem necessárias (e.g., nome e número dos alunos que o desenvolveram, limitações na implementação do projeto, interpretações em pontos que considerem ambíguos), num ficheiro comprimido no **formato ZIP**. O nome do ficheiro será **`SO-XXX-p1.zip`** (`XXX` é o número do grupo).
2. Submeter o ficheiro `SO-XXX-p1.zip` na página da disciplina no moodle da FCUL, utilizando o recurso disponibilizado para tal. Apenas um dos elementos do grupo deve submeter, considerando que será escolhida aleatoriamente uma submissão no caso de existirem várias.

Na entrega do trabalho, é ainda necessário ter em conta que:

- se não for incluído um `makefile`, se o mesmo não compilar os ficheiros fonte ou se houver erros de compilação (isto é, se não forem criados os ficheiros objeto e executáveis), o trabalho do grupo pode ser considerado nulo. Por isso garantam que o código submetido compila corretamente com o `make`.
- Todos os ficheiros entregues devem começar com um cabeçalho com três ou quatro linhas de comentários indicando o número do grupo, o nome e o número dos seus elementos.
- Os programas são testados no ambiente Linux instalado nos laboratórios de aulas, pelo que se recomenda que os alunos desenvolvam e testem os seus programas neste ambiente. A imagem Linux instalada nos laboratórios pode ser descarregada de https://admin.di.fc.ul.pt/downloads/.

Se não se verificar algum destes requisitos o trabalho é considerado não entregue.
Não serão aceites trabalhos entregues por email nem por qualquer outro meio não definido nesta secção.

7 Prazo de Entrega

O trabalho deve ser entregue até **às 23:59h** do dia **06 de abril de 2025** (domingo). Após esta data, a submissão do trabalho através do Moodle deixará de estar disponível.

8 Avaliação dos Trabalhos

A avaliação do trabalho será realizada:

(1) pelos alunos, pelo preenchimento de um formulário de contribuição de cada aluno no desenvolvimento do projeto. O formulário será disponibilizado no Moodle e preenchido após a entrega do projeto.
(2) pelo corpo docente sobre um conjunto de testes.

Para além dos testes a efetuar, os seguintes parâmetros serão tidos em conta: funcionalidade, estrutura, desempenho, algoritmia, comentários, clareza do código, validação dos parâmetros de entrada e tratamento de erros.

9 Divulgação dos Resultados

A data prevista para a divulgação dos resultados da avaliação dos trabalhos é 06 de maio de 2025.

10 Plágios

Não é permitido aos alunos partilharem códigos com soluções, ainda que parciais, de nenhuma parte do projeto com outros alunos (nem através do Fórum da disciplina, nem por qualquer outro meio). Além disso, todos os códigos serão testados por um verificador de plágio. Caso alguma irregularidade seja encontrada, os projetos de todos os alunos envolvidos serão anulados e o caso será reportado aos órgãos responsáveis em Ciências@ULisboa.

Chamamos a atenção para o facto das plataformas generativas baseadas em Inteligência Artificial (e.g., o ChatGPT e o GitHub Co-Pilot) gerarem um número limitado de soluções diferentes para o mesmo problema, as quais podem ainda incluir padrões característicos deste tipo de ferramenta e que podem vir a ser detetáveis pelos verificadores de plágio. Desta forma, recomendamos fortemente que os alunos não submetam trechos de código gerados por este tipo de ferramenta a fim de evitar riscos desnecessários.

Por fim, é responsabilidade de cada aluno garantir que a sua *home*, as suas diretorias e os seus ficheiros de código estão protegidos contra a leitura de outras pessoas (que não o utilizador dono dos mesmos). Por exemplo, se os ficheiros estiverem gravados na sua área de aluno nos servidores de Ciências@ULisboa, então todos os itens mencionados anteriormente devem ter as permissões de acesso `700`. Se os ficheiros estiverem no GitHub, garantam que o conteúdo do vosso repositório não esteja visível publicamente. Caso contrário, a sua participação no plágio será considerada ativa.