Trabalho de Projeto

2ª Fase

1 Introdução

A leitura prévia do enunciado da fase anterior é fundamental e deve ser considerada como uma introdução a este enunciado. O objetivo geral desta fase é 1) a sincronização entre processos no acesso à memória, 2) a escrita e leitura de ficheiros, 3) a gestão de tempo e 4) a captura e tratamento de alarmes e sinais.

2 Descrição Geral

A 2ª Fase 2 do Projeto irá partir do trabalho realizado na 1ª Fase, alterando alguns componentes e adicionando outros novos. Os principais componentes a desenvolver nesta fase são:

- **Sincronização de processos no acesso à memória** – com funções para a criação de semáforos e a sua utilização segundo o modelo produtor/consumidor. Para isso será fornecido o ficheiro `synchronization.h` e os alunos terão de desenvolver o código fonte correspondente (`synchronization.c`).
- **Ficheiro de Entrada** – a leitura dos 5 argumentos iniciais do ***SOChain*** (e.g., `init_balance` ou `n_wallets`) deixará de ser feita pela linha de comandos e passará a ser feita através de um ficheiro de entrada, onde em cada linha é guardado um argumento diferente. O nome deste ficheiro é passado pelos argumentos da linha de comandos (e.g., `$./SOChain args.txt ...`).
- **Temporização** – o `main`, `Wallets` e `Servers` passam a registar, nas operações que recebem, o instante temporal em que as processam.
- **Ficheiro de Log** – As 7 operações introduzidas pelo utilizador, depois de executadas, passam também a ser guardadas num ficheiro de log, que servirá como histórico de utilização do ***SOChain***.
- **Alarmes** – o `main` deve imprimir periodicamente no ecrã o estado atual de todos os pedidos. O período `T` deve estar escrito no ficheiro `settings.txt`, e o nome do ficheiro passado pelos argumentos da linha de comandos (e.g., `$./SOChain args.txt settings.txt`).
- **Sinais** – é necessário efetuar a captura do sinal de interrupção (`CTRL+C`) de programa para efetuar o fecho normal do ***SOChain*** tal como acontece quando utilizador escreve a opção *end.*
- **Ficheiro de Estatísticas** – as estatísticas finais do ***SOChain***, para além de incluírem estatísticas de processos (i.e., `main`, `Wallets` e `Servers`), passam também a incluir estatísticas das operações. Quando o ***SOChain*** termina, estas estatísticas são impressas no ecrã e escritas num ficheiro de estatísticas.

3 Descrição Específica

Nesta fase do projeto, os alunos terão de implementar:

- as funções da interface já existente `synchronization.h`.
- desenvolver 5 novas interfaces (e ficheiros de código fonte correspondentes): `csettings.h`, `clog.h`, `ctime.h`, `csignal.h` e `cstats.h`.
- caso desenvolvam funções auxiliares em algumas unidades de código, podem organizá-las em ficheiros de interfaces adicionando o sufixo `X-private.h` e código-fonte, cujas implementações deverão ser incorporadas também nos respetivos `X.c`.
- alterações que acharem necessárias ao código da 1ª Fase do seu projeto.

Não alterar nenhuma interface desenvolvida e fornecida pelos professores.

3.1 Sincronização

O ficheiro `synchronization.h` será fornecido pelos docentes e não poderá ser alterado pelos alunos. Neste caso, cabe aos alunos desenvolver apenas o código fonte `synchronization.c`. Junto com o ficheiro `synchronization.h`, são dadas também as interfaces da 1ª fase atualizadas para o tratamento da sincronização.

As novas funções e as funções modificadas são especificadas abaixo.

- Novas funções inseridas em `main.h`, as quais os alunos deverão implementar em `main.c`:

```
/* Função que inicializa os semáforos da estrutura semaphores. Semáforos
* *_full devem ser inicializados com valor 0, semáforos *_empty com valor
* igual ao tamanho dos buffers de memória partilhada
* Para tal pode ser usada a primitiva POSIX sem_init().*/
struct semaphores* create_semaphores();

/* Função que acorda todos os processos adormecidos em semáforos, para que
* estes percebam que foi dada ordem de terminação do programa.*/
void wakeup_processes(struct info_container * info);

/* Função que liberta todos os semáforos da estrutura semaphores. */
void destroy_semaphores(struct semaphores* sems);
```

De forma geral, `main.c` passa a estrutura `info_container` para as suas funções. Assim sendo, os apontadores para semáforos são agora parte nova desta estrutura.

```
struct info_container {
      float  init_balance; //saldo inicial
      int    n_wallets;
      ...
      int    *terminate;
      struct semaphores* sems; //novo elemento na estrutura
};
```

Para auxiliar o desenvolvimento desta fase do projeto, é também fornecida uma função `main` (atualizada) que os alunos podem usar:

```
int main(int argc, char *argv[]) {
      //init data structures (as in P1)
      //namely mem shared and dynamic memory
      ...
      //New: create semaphore data structure and note it in the info struct.
      struct semaphores* sems = create_semaphores();
      info->sems = sems;
      //register a Signal handler for CTRL+C
      ...
      //execute main Loop (as in P1)
      ...
      //release all memory before terminating:
      //1. free shared_mem (as in P1)
      ...
      //2. destroy semaphores:
      destroy_semaphores(info->sems);
      //3. Free remaining dynamic memory (as in P1)
      ...
}

struct semaphores* sems create_semaphores(){
      create_dynamic_memory(...);
      sems->main_wallets = create_dynamic_memory(...);
      sems->wallets_servers = create_dynamic_memory(...);
      sems->servers_main = create_dynamic_memory(...);
      ...
      //Init semaphores as needed
      ...
      return sems;
}

void wakeup_processes(struct info_container * info){
      ...
}
```

```
void destroy_semaphores(struct semaphores* sems){
      destroy_dynamic_memory(sems->main_wallets);
      destroy_dynamic_memory(sems->wallets_servers);
      destroy_dynamic_memory(sems->servers_main);
      destroy_dynamic_memory(sems);
}
```

3.2 Ficheiros de Entrada

Ficheiro `args.txt` – Este ficheiro obrigatório irá substituir a atual leitura de argumentos da linha de comandos, sendo lido no início da execução do ***SOChain***. Em cada linha deste ficheiro é listado um dos argumentos iniciais de ***SOChain***, seguindo o seguinte formato

```
init_balance            // saldo incial das carteiras
n_wallets               //nº de carteiras
n_servers               //nº de servidores
buff_size               //tamanho dos buffers
max_txs                 //nº máximo de transações
```

Ficheiro `settings.txt` – Este ficheiro obrigatório deve conter três parâmetros, um por linha, nomeadamente:

```
log_filename            //nome do ficheiro log a ser usado pelo SOChain
statistics_filename     //nome do ficheiro de estatísticas
                        // a ser gerado no fim da execução
period                  //periodo de impressão das estatísticas no ecrã.
                        // Usar 0 para desativar impressão
```

Estes módulos de leitura dos ficheiros de entradas devem ser desenvolvidos no ficheiro `csettings.h`, sendo também necessário efetuar a ligação entre este e os módulos da 1ª fase do projeto.

3.3 Temporização

O módulo de temporização é outro dos novos módulos a desenvolver. Nomeadamente, os todos os processos do ***SOChain*** passam a usar funções de temporização para registar o instante no tempo em que alteram uma transação. Para guardar estes tempos, deve ser adicionado um novo campo na estrutura chamado `change_time`:

```
#include "ctime.h"
struct transaction {
      int id; //ID da transação
      int src_id;
      ...
      int wallet_signature;
      int server_signature;
      struct timestamps change_time; //novo campo que contêm três timestamps
};
```

Este módulo deve ser desenvolvido no ficheiro `ctime.h`, fazendo também as alterações e ligações necessárias com os módulos da 1ª fase do projeto. O registo de tempos deve ser feito usando a função `clock_gettime` da biblioteca standard `time.h`.

3.4 Ficheiro de Log

A `main` (processo principal) passa a guardar um registo de todas as operações executadas pelo utilizador. Tal registo é guardado num ficheiro de log, cujo nome a usar deverá provir de `settings.txt` (argumento `log_filename`). As operações do utilizador ("`bal 10`", "`rcp 20`", *etc.*) são guardadas no seguinte formato:

```
time operation
```

onde `time` é o instante em que a operação foi feita pelo utilizador, indicando o ano, mês, dia, hora, minuto, segundo e milissegundos, como exemplificado abaixo. Para indicarem estes tempos devem usar as funções `clock_gettime` e `localtime` da biblioteca `time.h`:

```
20250131 14:53:19.943 bal 10
20250131 14:53:30.572 rcp 2
20220201 23:14:21.326 help
20220221 14:55:42.682 end
```

Este módulo deve ser desenvolvido no ficheiro `clog.h`, sendo também necessário efetuar as devidas alterações e ligações com os módulos da 1ª fase do projeto.

3.5 Alarmes

Neste módulo, pretende-se armar um alarme que periodicamente verifique e escreva para o ecrã o estado atual de todas as transações, incluindo: (*i*) as que já foram finalizadas e (*ii*) os que ainda estão em andamento. O estado duma transação é escrito segundo o seguinte formato:

```
id elapsed_time
```

Onde `id` é o identificador da transação e `elapsed_time` representa o número de segundos (apresentar 3 casas decimais) que a transação passa no sistema. O tempo começa a contar desde que é registada no *buffer* pela `main`, e termina quando a `main` pede o seu recibo.

```
3 6.345
4 12.345
5 12.345
6 3.122
7 0.022
```

A periodicidade do alarme é o valor do `period` passado no ficheiro de entrada `settings.txt`. O alarme deverá assim ser armado de `<period>` em `<period>` segundos. Este módulo deve ser desenvolvido no ficheiro `csignal.h`, fazendo também as alterações e ligações necessárias com os módulos da 1ª fase do projeto.

3.6 Sinais

Neste módulo, pretende-se capturar o sinal de interrupção de programa (`SIGINT` – ativado pela combinação de teclas `CTRL+C`) de forma a libertar todos os recursos do ***SOChain*** e efetuar um fecho normal do programa. Assim, ao capturar este sinal, deve-se proceder à invocação da função `end_execution` no processo pai, e garantir que os processos filhos fecham corretamente. Apenas o processo pai deve capturar este sinal, enquanto os processos filhos devem ignorá-lo e esperar que o tratamento do sinal no processo pai modifique a variável partilhada `terminate`. Este módulo deve ser desenvolvido no ficheiro `csignal.h`, fazendo também as alterações e ligações necessárias com os módulos da 1ª fase do projeto.

3.7 Ficheiro de Estatísticas

Por fim, as estatísticas finais do ***SOChain*** passam a ser escritas para um ficheiro de estatísticas, que irá incluir tanto as estatísticas dos vários processos como das inúmeras transações. O nome do ficheiro de estatísticas é passado dentro do ficheiro `settings.txt` (argumento `statistics_filename`).

Este ficheiro terá o formato exemplificado abaixo:

```
Process Statistics:
Main has PID [1120]
There were 2 Wallets, PIDs [1121,1122]
There were 3 Servers, PIDs [1123,1124,1125]
Main received 16 transaction(s)!
Wallet #0 signed 5 transaction(s)!
Wallet #1 signed 2 transaction(s)!
Server #0 processed 0 transaction(s)!
Server #1 processed 1 transaction(s)!
Server #2 processed 4 transaction(s)!
Main read 5 receipts.
Final Balances [1.23,2.13] SOT
```

Este módulo deve ser desenvolvido no ficheiro `cstats.h`, sendo também necessário fazer as devidas alterações e ligações com os módulos da 1ª fase do projeto.

4 Estrutura do Projeto e makefile

O projeto deve ser organizado de acordo com a estrutura da 1ª fase do projeto, onde os ficheiros `.h` serão incluídos na diretoria `inc/` e os ficheiros `.c` serão incluídos na diretoria `src/`. O `makefile` necessário para a execução do comando `make` será desenvolvido pelos alunos e colocado na raiz do diretório `SOChain`. Deste modo, os alunos devem atualizar o ficheiro `makefile` da fase anterior do projeto com as instruções necessárias da 2ª fase, por forma que o executável seja gerado a partir da execução do comando `make`.

5 Entrega

A entrega da 2ª 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 usada na 1ª fase do projeto, bem como um ficheiro `README` onde os alunos podem incluir informações que julguem necessárias (p. ex.: nome e número dos alunos que o desenvolveram, limitações na implementação do projeto, etc.), num ficheiro comprimido no formato ZIP. O nome do ficheiro será **`SO-XXX-p2.zip`** (**`XXX`** é o número do grupo).
2. Submeter o ficheiro `SO-XXX-p2.zip` na página da disciplina no moodle da FCUL, utilizando o recurso disponibilizado para tal. Apenas um dos elementos do grupo deve submeter, evitando dessa forma que seja escolhida aleatoriamente uma submissão no caso de existirem várias.

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

1. Se não for incluído um `makefile`, **o trabalho é considerado nulo**.
2. Se o mesmo não compilar os ficheiros fonte**, o trabalho é considerado nulo**.
3. Se houver erros de compilação**, o trabalho é considerado nulo**.
4. Todos os ficheiros entregues devem começar com um cabeçalho com três ou quatro linhas de comentários:
   a. indicando o número do grupo,
   b. o nome e número dos seus elementos.
   c. Se não se verificar algum destes requisitos o ficheiro é considerado não entregue.
5. 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 nesse ambiente.
6. 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.

6 Prazo de Entrega

O trabalho deve ser entregue até às 23:59h do dia **25 de maio de 2025 (domingo)**. Após esta data, a submissão do trabalho através do Moodle deixará de ser permitida.

7 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 também terá de ser preenchido até **ao prazo de entrega**.

(2) por discussão dos trabalhos dos alunos com os docentes. As discussões serão realizadas presencialmente, entre 26 e 30 de maio de 2025. Todos os elementos do grupo terão de comparecer à avaliação e a avaliação é feita individualmente. Deste modo, cada elemento do grupo deve estar preparado para responder a qualquer questão relacionada com os trabalhos e com a matéria das aulas teórico-práticas.

(3) 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.

8 Divulgação dos Resultados

A data prevista da divulgação dos resultados da avaliação dos trabalhos é 18 de junho de 2025.

9 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.