Ciências ULisboa | Informática

2022/23

Sistemas Operativos

Projeto – Fase 2

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 a sincronização entre processos no acesso à memória, a escrita e leitura de ficheiros, a gestão de tempo, e a captura e tratamento de alarmes e sinais.

2. Descrição Geral

A fase 2 do projeto irá partir do trabalho realizado na fase 1, 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 argumentos iniciais do ***AdmPor*** (p.ex.: número máximo de pedidos, tamanho dos buffers, número de intermediários, número de empresas, número de clientes, entre outros) 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 (p.ex.: `$./admpor config.txt`).
- **Temporização** – os clientes, intermediários e empresas passam a registar, nas operações que recebem, o instante temporal em que as processam.
- **Ficheiro de Log** – todas as operações introduzidas pelo utilizador, depois de executadas, passam também a ser guardadas num ficheiro de log, que servirá como histórico de utilização do ***AdmPor***.
- **Alarmes** – é definido um novo parâmetro de entrada (passado no ficheiro de entrada) que determina o intervalo de tempo para um alarme que, quando ativado, imprime para o ecrã o estado atual de todos os pedidos.
- **Sinais** – é necessário efetuar a captura do sinal de interrupção de programa para efetuar o fecho normal do ***AdmPor*** e a libertação de todos os semáforos e zonas de memória.
- **Ficheiro de Estatísticas** – as estatísticas finais do ***AdmPor***, para além de incluírem estatísticas de processos (i.e., clientes, intermediários e empresas), passam também a incluir estatísticas das operações. Quando o ***AdmPor*** 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 *synchronization.h*, desenvolver as suas próprias interfaces (ficheiros *.h*) e ficheiros de código fonte correspondentes (ficheiros *.c*) para as restantes componentes do trabalho, e efetuar as alterações que acharem necessárias ao código da fase 1 do seu projeto (excetuando as interfaces) de forma a concretizar os novos objetivos. Nomeadamente, os alunos deverão desenvolver 5 novas interfaces (e ficheiros de código fonte correspondentes): ***configuration.h***, ***log.h***, ***aptime.h, apsignal.h*** e ***stats.h***.

3.1. Interfaces de sincronização e da 1ª fase do projeto e Ficheiro Main

A exceção é o ficheiro *synchronization.h* que, conforme descrito na Seção 2, será fornecido pelos docentes e não poderá ser alterado pelos alunos. Neste caso, cabe aos alunos desenvolver apenas o código fonte *synchronization.c*.

Os alunos podem criar o ficheiro *synchronization-private.h* se considerarem necessário para o seu trabalho. Já os ficheiros **-private.h* que os alunos desenvolveram na fase 1 do projeto, terão de migrar para a fase 2.

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, e os *_mutex com
* valor igual a 1. Para tal pode ser usada a função semaphore_create.*/
void create_semaphores(struct main_data* data, struct semaphores* sems);

/* Função que acorda todos os processos adormecidos em semáforos, para que
* estes percebam que foi dada ordem de terminação do programa. Para tal,
* pode ser usada a função produce_end sobre todos os conjuntos de semáforos
* onde possam estar processos adormecidos e um número de vezes igual ao
* máximo de processos que possam lá estar.*/
void wakeup_processes(struct main_data* data, struct semaphores* sems);

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

- Funções modificadas em *main.h*, as quais os alunos deverão reprogramar tendo em conta o novo parâmetro de entrada (em **negrito**):

```
void launch_processes(struct comm_buffers* buffers, struct main_data* data,
struct semaphores* sems);

void user_interaction(struct comm_buffers* buffers, struct main_data* data,
struct semaphores* sems);

void create_request(int* op_counter, struct comm_buffers* buffers, struct
main_data* data, struct semaphores* sems);

void read_status(struct main_data* data, struct semaphores* sems);

void stop_execution(struct main_data* data, struct comm_buffers* buffers,
struct semaphores* sems);
```

- Funções modificadas em *process.h*, as quais os alunos deverão reprogramar tendo em conta o novo parâmetro de entrada (em **negrito**):

```
int launch_client(int client_id, struct comm_buffers* buffers, struct
main_data* data, struct semaphores* sems);

int launch_interm(int interm_id, struct comm_buffers* buffers, struct
main_data* data, struct semaphores* sems);

int launch_enterp(int enterp_id, struct comm_buffers* buffers, struct
main_data* data, struct semaphores* sems);
```

- Funções modificadas em *intermediary.h*, as quais os alunos deverão reprogramar tendo em conta o novo parâmetro de entrada (em **negrito**):

```
int execute_intermediary(int interm_id, struct comm_buffers* buffers,
struct main_data* data, struct semaphores* sems);

void intermediary_receive_operation(struct operation* op, struct
comm_buffers* buffers, struct main_data* data, struct semaphores* sems);

void intermediary_process_operation(struct operation* op, int interm_id,
struct main_data* data, int* counter, struct semaphores* sems);

void intermediary_send_answer(struct operation* op, struct comm_buffers*
buffers, struct main_data* data, struct semaphores* sems);
```

- Funções modificadas em *enterprise.h*, as quais os alunos deverão reprogramar tendo em conta o novo parâmetro de entrada (em **negrito**):

```
int execute_enterprise(int enterp_id, struct comm_buffers* buffers, struct
main_data* data, struct semaphores* sems);

void enterprise_receive_operation(struct operation* op, int enterp_id,
struct comm_buffers* buffers, struct main_data* data, struct semaphores*
sems);

void enterprise_process_operation(struct operation* op, int enterp_id,
struct main_data* data, int* counter, struct semaphores* sems);
```

- Funções modificadas em *client.h*, as quais os alunos deverão reprogramar tendo em conta o novo parâmetro de entrada (em **negrito**):

```
int execute_client(int client_id, struct comm_buffers* buffers, struct
main_data* data, struct semaphores* sems);

void client_get_operation(struct operation* op, int client_id, struct
comm_buffers* buffers, struct main_data* data, struct semaphores* sems);

void client_process_operation(struct operation* op, int client_id, struct
main_data* data, int* counter, struct semaphores* sems);

void client_send_operation(struct operation* op, struct comm_buffers*
buffers, struct main_data* data, struct semaphores* sems);
```

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
      struct main_data* data = create_dynamic_memory(sizeof(struct
main_data));
      struct comm_buffers* buffers = create_dynamic_memory(sizeof(struct
comm_buffers));
      buffers->main_client = create_dynamic_memory(sizeof(struct
rnd_access_buffer));
      buffers->client_interm = create_dynamic_memory(sizeof(struct
circular_buffer));
      buffers-> interm_enterp = create_dynamic_memory(sizeof(struct
rnd_access_buffer));

      // init semaphore data structure
      struct semaphores* sems = create_dynamic_memory(sizeof(struct
semaphores));
      sems->main_client = create_dynamic_memory(sizeof(struct prodcons));
      sems->client_interm = create_dynamic_memory(sizeof(struct prodcons));
      sems->interm_enterp = create_dynamic_memory(sizeof(struct prodcons));

      //execute main code
      main_args(argc, argv, data);
      create_dynamic_memory_buffers(data);
      create_shared_memory_buffers(data, buffers);
      create_semaphores(data, sems);
      launch_processes(buffers, data, sems);
      user_interaction(buffers, data, sems);

      //release memory before terminating
      destroy_dynamic_memory(data);
      destroy_dynamic_memory(buffers->main_client);
      destroy_dynamic_memory(buffers->client_interm);
      destroy_dynamic_memory(buffers->interm_enterp);
      destroy_dynamic_memory(buffers);
      destroy_dynamic_memory(sems->main_client);
      destroy_dynamic_memory(sems->client_interm);
      destroy_dynamic_memory(sems->interm_enterp);
      destroy_dynamic_memory(sems);
}
```

3.2. Ficheiro de Entrada

Este ficheiro irá substituir a atual leitura de argumentos da linha de comandos, sendo lido no início da execução do ***AdmPor***. Em cada linha deste ficheiro é listado um dos argumentos iniciais de ***AdmPor***, seguindo o seguinte formato:

```
max_ops                    //nº máximo de operações
buffers_size               //tamanho dos buffers
n_clients                  //nº de clientes
n_intermediaries           //nº de intermediários
n_enterprises              //nº de empresas
log_filename               //nome do ficheiro de log
statistics_filename        //nome do ficheiro de estatísticas
alarm_time                 //temporização para o alarme
```

Atenção aos novos argumentos `log_filename`, `statistics_filename`, e `alarm_time`. O nome do ficheiro de entrada é passado ao ***AdmPor*** através dos argumentos da linha de comandos, e passa a ser o único argumento introduzido dessa forma.

Este módulo de leitura do ficheiro de entrada deve ser desenvolvido no ficheiro ***configuration.h***, sendo também necessário efetuar a ligação entre este e os módulos da fase 1 do projeto.

3.3. Temporização

O módulo de temporização é outro dos novos módulos a desenvolver. Nomeadamente, os clientes, intermediários e empresas passam a usar funções de temporização para registar o instante no tempo em que processam cada operação recebida. Adicionalmente, a `main` regista o instante em que uma operação é inicializada (i.e., quando o utilizador cria um pedido).

Para guardar estes tempos, devem ser adicionados novos campos na estrutura *operation*:

```
struct operation {
      ...                                   //manter os campos da 1ª fase
      struct timespec start_time;           //quando o pedido foi criado
      struct timespec client_time;          //quando o cliente recebeu
                                            //o pedido
      struct timespec intermed_time;        //quando o intermediário recebeu
                                            //o pedido
      struct timespec enterp_time;          //quando a empresa recebeu
                                            //o pedido
};
```

O registo de tempos deve ser feito usando a função *clock_gettime* da biblioteca *time.h*.

Este módulo deve ser desenvolvido no ficheiro ***aptime.h***, fazendo também as alterações e ligações necessárias com os módulos da fase 1 do projeto.

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 é passado como um dos argumentos do ficheiro de entrada (argumento `log_filename`). As operações do utilizador (*op*, *status*, *stop e help*) são guardados no seguinte formato:

```
time operation argument
```

onde `time` é o instante em que a operação foi feita pelo utilizador, indicando o ano, mês, dia, hora, minuto, segundo e milissegundo. Para indicarem estes tempos podem usar as funções *clock_gettime* e *localtime* da biblioteca *time.h;* `operation` é a operação (*op*, *status*, *stop e help*); e `argument` é o argumento da operação (caso exista) passado pelo utilizador. Segue um exemplo de ficheiro de log:

```
2022-2-31 14:53:19.943 op 1 2
2022-2-31 14:53:30.572 status 0
2022-2-31 14:54:21.326 help
2022-2-31 14:55:42.682 stop
```

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

3.5. Alarmes

Neste módulo, pretende-se armar um alarme que em certos intervalos de tempo (p. ex.: de X em X segundos, onde X é o valor do `alarm_time` passado no ficheiro de entrada) verifique e escreva para o ecrã o estado atual de todos os pedidos, incluindo: (*i*) os que já foram finalizados

pelas empresas e (*ii*) os que ainda estão em andamento ou agendados. O estado dos pedidos é escrito segundo o seguinte formato:

```
op status start_time receiving_client client_time receiving_intermed
intermed_time receiving_enterprise enterprise_time
```

Aqui, os tempos devem ser apresentados em segundos e no formato *raw*, ou seja, sem aplicar a função *localtime*. Um pedido é reportado com status 'M', 'C', 'I', 'A', ou 'E', consoante o seu estado (ver o enunciado da fase 1).

Por exemplo, para um *max_ops* = 3, o output do alarme poderia ser:

```
op:0 status:E start_time: 1617202725 client:1 client_time: 1617202726
intermediary:1 intermediary_time: 1617202726 enterprise:2 enterprise_time:
1617202728
op:1 status:C start_time: 1617202729 client:1 client_time: 1617202731
op:2 status:M start_time: 1617202732
```

Este módulo deve ser desenvolvido no ficheiro ***apsignal.h***, fazendo também as alterações e ligações necessárias com os módulos da fase 1 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 ***AdmPor*** e efetuar um fecho normal do programa. Assim, ao capturar este sinal, deve-se proceder à invocação da função *stop_execution* (para tal, os alunos poderão ter que passar as variáveis *main_data*, *communication_buffers* e *semaphores* para variáveis globais) e garantir que tanto o processo pai como os processos filhos (clientes, intermediários e empresas) capturam o sinal.

Este módulo deve ser desenvolvido no ficheiro ***apsignal.h***, fazendo também as alterações e ligações necessárias com os módulos da fase 1 do projeto.

3.7. Ficheiro de estatísticas

Por fim, as estatísticas finais do ***AdmPor*** passam a ser escritas para um ficheiro de estatísticas, que irá incluir tanto as estatísticas dos vários processos (clientes, intermediários e empresas) como dos vários pedidos. O nome do ficheiro de estatísticas é passado como um dos argumentos do ficheiro de entrada (argumento `statistics_filename`).

Este ficheiro terá o seguinte formato:

```
Process Statistics:
      Client ... received ... operation(s)!
      ...
      Intermediary ... prepared ... operation(s)!
      ...
      Enterprise ... executed ... operation(s)!
      ...

Request Statistics:
Request: …
Status: …
Client id: …
Intermediary id: …
Enterprise id: …
Start time: …
Client time: …
```

```
Intermediary time: …
Enterprise time: …
Total Time: …

Request: …
...
```

Os tempos `"Start time", "Client time", "Intermediary time" e "Enterprise time"` indicam o ano, mês, dia, hora, minuto, segundo e milissegundos. Para indicarem estes tempos podem usar a função *localtime* da biblioteca *time.h.*

O tempo `Total Time` é a diferença entre o `"Enterprise time"` e o `"Start time"`, sendo apresentado em segundos (com 3 casa decimais de milissegundos).

Segue um exemplo concreto:

```
Process Statistics:
      Client 1 received 1 operation(s)!
      Intermediary 0 prepared 1 operation(s)!
      Enterprise 2 executed 1 operation(s)!

Request Statistics:
Request: 0
Status: E
Intermediary id: 0
Enterprise id: 2
Client id: 1
Start time: 2023-4-14 14:53:19.806
Client time: 2023-4-14 14:53:20.372
Intermediary time: 2023-4-14 14:53:20.534
Enterprise time: 2023-4-14 14:53:22.575
Total Time: 2.769
```

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

4. Estrutura do projeto e makefile

O projeto deve ser organizado de acordo com a estrutura da fase 1 do projeto, onde os ficheiros `.h` serão incluídos na directoria `include` e os ficheiros *.c* serão incluídos na directoria `src`.

O `makefile` necessário para a execução do comando `make` será desenvolvido pelos alunos e colocado na raiz do diretório `ADMPOR`. Deste modo, os alunos devem atualizar o ficheiro `makefile` da fase anterior do projeto com as instruções necessárias da fase 2, por forma que o executável `admpor` seja gerado a partir da execução do comando `make`.

5. Entrega

A entrega da fase 2 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 fase 1 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 **`grupoXX-projeto2.zip`** (XX é o número do grupo).

2. Submeter o ficheiro **`grupoXX-projeto2.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` e (2) se o mesmo não compilar os ficheiros fonte, ou (3) se houver erros de compilação (isto é, se não forem criados os ficheiros objeto e executáveis), o trabalho é considerado nulo**.
- Todos os ficheiros entregues devem começar com um cabeçalho com três ou quatro linhas de comentários indicando o número do grupo, o nome e 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 nesse ambiente. A imagem Linux instalada nos laboratórios pode ser descarregada de https://admin.di.fc.ul.pt/importacao-da-imagem-para-virtualbox-tutorial/

Se não se verificar algum destes requisitos o trabalho é considerado não entregue.

6. Prazo de entrega

O trabalho deve ser entregue até dia **21 de maio de 2023 (domingo) às 23:59h**. Após esta data, a submissão do trabalho através do Moodle deixará de ser permitida.

Não serão aceites trabalhos entregues por mail nem por qualquer outro meio não definido nesta secção.

7. Avaliação dos trabalhos

A avaliação do trabalho será realizada:

(1) pelos alunos, pelo preenchimento obrigatório do formulário de contribuição de cada aluno no desenvolvimento do projeto. O formulário será disponibilizado no Moodle na semana anterior à entrega do projeto e também terá de ser preenchido até **21 de maio de 2023, às 23:59h**.

(2) por discussão dos trabalhos dos alunos com os docentes. As discussões serão realizadas presencialmente, entre 22 e 26 de maio de 2023. 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 avaliados: 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 12 de junho de 2023.

9. Plágio

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 na Ciências@ULisboa.

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 da Ciências@ULisboa, então todos os itens mencionados anteriormente devem ter as permissões de acesso 700.