Ciências ULisboa | Informática

2020/21

Sistemas Operativos

Projeto – Fase 1

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 para simular um sistema de pedidos e respostas genérico, com três tipos de participantes diferentes: clientes, *proxies* e servidores. Esta aplicação, chamada ***SOVACCINES***, simula o processo de aquisição e distribuição de vacinas, onde os clientes são as instalações de saúde e farmacêuticas (ex., Centro de Saúde de Alvalade), os *proxies* são os serviços de distribuição (ex., Alliance Healthcare, S.A.), e os servidores são os laboratórios farmacêuticos que produzem e fornecem as vacinas (ex., Johnson & Johnson, Lda). Uma interação entre cliente-servidor, através do proxy, seria, por exemplo, o Centro de Saúde de Alvalade efetuar uma requisição (pedido) de 150 vacinas (ex., vacina da gripe) à Alliance Healthcare, S.A.; este último recebe, processa e reencaminha o pedido para o laboratório farmacêutico Johnson & Johnson, Lda que fornece aquela vacina, após verificar que o laboratório o pode satisfazer; a Johnson & Johnson, Lda recebe a requisição da Alliance Healthcare, S.A. e expede-a, enviando as vacinas diretamente para o Centro de Saúde de Alvalade. A Figura 1 ilustra a visão geral do sistema e as interações entre os participantes.

O ***SOVACCINES*** será realizado em 2 fases. A primeira fase tem como objetivo fundamental a familiarização com a linguagem C, a criação do ficheiro `makefile` (para compilação e manutenção do projeto), a gestão de processos, a alocação de memória, e sincronização entre processos no acesso a memória. Neste primeiro enunciado é feita uma apresentação geral da ***SOVACCINES*** juntamente com informação específica de suporte à realização da primeira fase. Na fase seguinte será disponibilizado um novo enunciado que complementará a informação contida neste enunciado.

2. Funcionamento Geral

Como referido acima, no modelo do sistema do ***SOVACCINES***, existem três tipos de participantes: clientes, proxies e servidores. Em cada execução do ***SOVACCINES*** pode haver um ou mais participantes de cada tipo, conforme configurado pelo utilizador do sistema. Para além desses três participantes, existe também o processo principal (processo pai), i.e., o *Main*, que gere todos os outros processos (filhos) e a interação com o utilizador. O Main oferece um menu com opções para que o utilizador possa interagir com ***SOVACCINES***.

O menu contém quatro opções:

(1) `op` – criar um pedido de aquisição de vacinas;
(2) `read` – consultar o estado de um dado pedido (especificado pelo utilizador);
(3) `stop` – terminar a execução do sistema ***SOVACCINES***;
(4) `help` – mostrar as informações de ajuda sobre as opções anteriores.

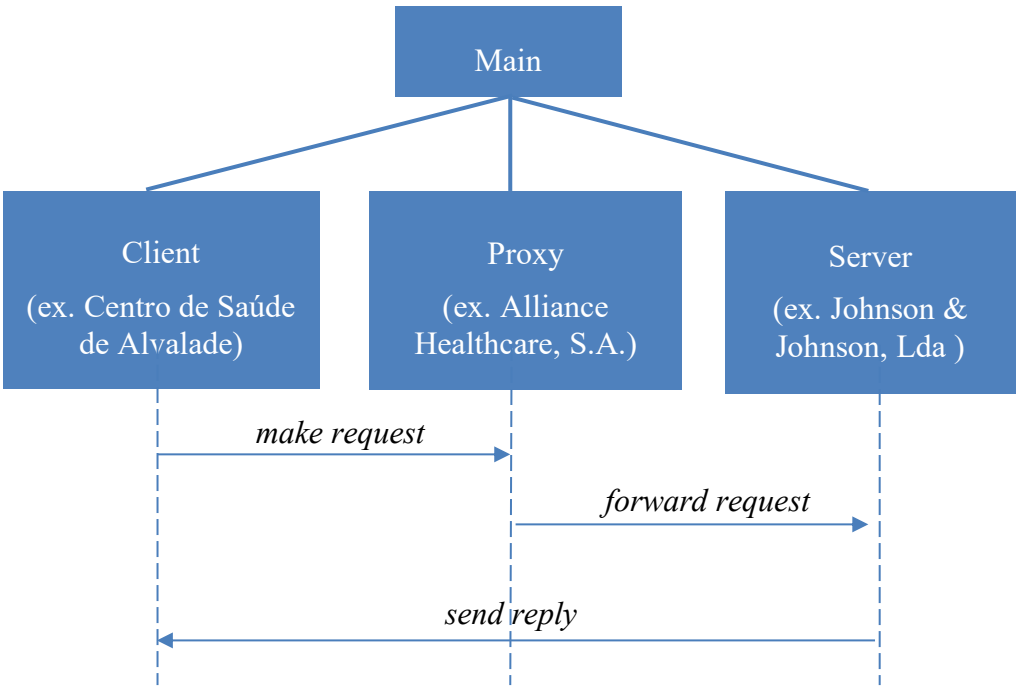

Figura 1: Visão geral do sistema SOVACCINES e interações entre os participantes.

Quando é criado um pedido (1), a *Main* cria uma operação representativa do pedido e a envia para ser processada pelos clientes. Um dos clientes irá processar o pedido e enviará o pedido processado para os proxies. Um dos proxies irá receber o pedido e o encaminhará para os servidores. Por sua vez, um dos servidores irá receber o pedido encaminhado e irá enviar uma resposta ao cliente. Por fim, um dos clientes recebe a resposta e guarda-a num histórico de operações executadas, o qual é mantido pelo processo principal. O modelo de interação entre os 3 participantes é o de produtor/consumidor e o critério de seleção de qual participante processa o pedido em curso é da responsabilidade do escalonador do CPU. Um pedido em curso transita entre três estados que indicam em que estado o seu processamento se encontra. Os estados são: 'C' - pedido processado por cliente; 'P' - pedido processado por proxy; 'S' - pedido processado por servidor, resultando o envio da resposta ao cliente e terminando o pedido.

A ação de consulta de estado de um pedido (2), por sua vez, verifica se a resposta a uma requisição para aquisição de vacinas já chegou e consulta as suas estatísticas, nomeadamente, o seu estado e os identificadores dos clientes, proxies e servidores que a atenderam.

Por fim, a ação para terminar a execução do sistema ***SOVACCINES*** (3) imprime as estatísticas finais do mesmo, como o número de pedidos processados por cada cliente, proxy e servidor, e termina o programa.

3. Descrição específica

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

- *Main* (main.c e main.h – main.c/h), módulo principal que gere os outros módulos e faz a interação com o utilizador.
- Gestão de processos (process.c/h), com funções para criar e destruir processos (ex., cliente, proxy e servidor).

- Gestão de memória (memory.c/h), com funções para criação de memória dinâmica e de memória partilhada, assim como funções para escrita em diferentes tipos de estruturas de dados (ex., buffers circulares e de acesso aleatório).
- Sincronização de processos (synchronization.c/h), com funções para criação de semáforos e sua utilização segundo o modelo produtor/consumidor.
- Clientes (client.c/h), com funções para receber pedidos da *Main* e encaminhar para os proxies, assim como para receber respostas de servidores e guardar as mesmas no histórico de operações executadas.
- Proxies (proxy.c/h), com funções para receber pedidos de clientes e encaminhar os mesmos para os servidores.
- Servidores (server.c/h), com funções para receber pedidos de proxies, processá-los e enviar as respostas para os clientes.

Para cada um destes módulos, é fornecido um ficheiro com extensão *.h* com os cabeçalhos das funções, e que **não pode ser alterado**. As concretizações das funções definidas nos ficheiros *X.h* (em que *X* é o nome do ficheiro) terão de ser desenvolvidas pelo grupo de trabalho nos ficheiros *X.c*, utilizando os algoritmos e métodos que o grupo achar convenientes. Se o grupo entender necessário, ou se for pedido, também pode criar um ficheiro de cabeçalho *X-private.h* para acrescentar outras definições, cujas implementações serão também incluídas no ficheiro *X.c*.

Juntamente com o enunciado da Fase 1 do projeto é disponibilizado, aos alunos, um ficheiro zip contendo todos os cabeçalhos definidos para os módulos acima apresentados e um conjunto de ficheiros "-private.h" sugerindo funções que podem ser úteis implementar pelo grupo de trabalho.

Para auxiliar no início do desenvolvimento do projeto, é também fornecida uma função main 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 communication_buffers* buffers =
create_dynamic_memory(sizeof(struct communication_buffers));
	buffers->main_cli = create_dynamic_memory(sizeof(struct
rnd_access_buffer));
	buffers->cli_prx = create_dynamic_memory(sizeof(struct
circular_buffer));
	buffers->prx_srv = create_dynamic_memory(sizeof(struct
rnd_access_buffer));
	buffers->srv_cli = create_dynamic_memory(sizeof(struct
circular_buffer));
	struct semaphores* sems = create_dynamic_memory(sizeof(struct
semaphores));
	sems->main_cli = create_dynamic_memory(sizeof(struct prodcons));
	sems->cli_prx = create_dynamic_memory(sizeof(struct prodcons));
	sems->prx_srv = create_dynamic_memory(sizeof(struct prodcons));
	sems->srv_cli = 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 final memory
        destroy_dynamic_memory(data);
        destroy_dynamic_memory(buffers->main_cli);
        destroy_dynamic_memory(buffers->cli_prx);
        destroy_dynamic_memory(buffers->prx_srv);
        destroy_dynamic_memory(buffers->srv_cli);
        destroy_dynamic_memory(buffers);
        destroy_dynamic_memory(sems->main_cli);
        destroy_dynamic_memory(sems->cli_prx);
        destroy_dynamic_memory(sems->prx_srv);
        destroy_dynamic_memory(sems->srv_cli);
        destroy_dynamic_memory(sems);
}
```

4. Estrutura do projeto e makefile

O projeto deve ser organizado segundo a seguinte estrutura:

`\SOVACCINES`

`\bin`

`\include`

`\obj`

`\src`

O diretório `bin` deverá conter o executável `sovaccines`. O diretório `include` 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 *-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 o código desenvolvidos pelos alunos.

O executável `sovaccines` será gerado a partir da execução do comando `make`. O `makefile` necessário para a execução do comando `make` será desenvolvido pelos alunos e deve ser colocado na raiz do diretório `SOVACCINES`.

5. Desenvolvimento e Testes

Esta fase do projeto inclui vários módulos e funções a desenvolver. Como tal, recomenda-se que os alunos as desenvolvam ao longo do semestre, após ter sido lecionada a matéria respetiva nas aulas TP, não devendo deixar a realização do mesmo para a última semana da data de entrega.

Para efeitos de teste, é fornecido aos alunos o executável `sovaccines_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, tanto o executável desenvolvido pelos alunos como o desenvolvido pelos professores devolvam resultados semelhantes.

Exemplo de utilização do executável `sovaccines` (e do `sovaccines_profs`):

```
$./sovaccines max_ops buffers_size n_clients n_proxies n_servers
```

onde `max_ops` é o número máximo de pedidos que podem ser criados, `buffers_size` é o tamanho máximo dos buffers, `n_clients` é o número máximo de clientes, `n_proxies` é o número máximo de proxies e `n_servers` é o número máximo de servidores.

Recomenda-se aos alunos que comecem os testes com um número reduzido de operações e processos (ex., 10 pedidos, 1 cliente, 1 proxy e 1 servidor). Caso o código funcione com este exemplo, então aumentem e testem com mais processos e operações.

Depois do arranque da aplicação, o utilizar deve interagir com o sistema por meio de um menu com as opções `op`, `read x` e `stop` que foram explicadas na Seção 2. Deve também ser possível por meio da ação `help` imprimir informações sobre as ações 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 Seção 4, **bem como um ficheiro README** onde os alunos podem incluir informações que julguem necessárias (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-projeto1.zip`** (XX é o número do grupo).
2. Submeter o ficheiro **`grupoXX-projeto1.zip`** na página da disciplina no moodle da FCUL, utilizando a atividade disponibilizada 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:

- **(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/

O prazo de entrega é dia 26/03/2021 até às 23h55min.

Após esta data, a submissão do trabalho através do Moodle deixará de ser permitida.