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 ***hOSpital*,** para simular a admissão e o atendimento de pacientes de um hospital. Esta aplicação permitirá aos pacientes fazerem a admissão com os rececionistas do hospital e realizarem as consultas com os médicos do mesmo. Esta aplicação envolverá múltiplos processos cooperativos para efetuar as suas operações. O fluxo de chamadas entre processos envolve: (i) a admissão do paciente junto a um rececionista do hospital, (ii) o encaminhamento dos pacientes para a sala de espera dos médicos e (iii) o atendimento médico dos pacientes. De forma a se poder aferir a qualidade do serviço prestado, são também registadas informações de progresso sob a forma de um log, que podem posteriormente ser analisadas.

O *hOSpital* será realizado em 2 fases. A primeira fase tem como objetivo fundamental a familiarização com a linguagem C, a criação do ficheiro `makefile` (para a compilação e manutenção do projeto), a gestão de processos e a alocação de memória. Neste primeiro enunciado é feita uma apresentação geral do *hOSpital* juntamente com a 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 *hOSpital*, existem três tipos de participantes: pacientes, rececionistas e médicos. Cada execução do *hOSpital* representa o funcionamento do hospital num dia, onde pode haver uma ou mais instâncias de cada tipo, conforme configurado pelo utilizador do sistema. Para além destes três participantes, existe também o processo principal (i.e., processo pai, a `Main`, que gere todos os outros processos filhos e a interação com o utilizador). A `Main` oferece um menu com opções para que o utilizador possa interagir com o *hOSpital*.

O menu contém quatro opções:

(1) `ad paciente médico` – o paciente `paciente` faz a admissão no hospital a indicar que pretende consultar com o médico `médico`. Como resultado, obtém uma senha de atendimento `id`;

(2) `info id` – consultar o estado da senha de admissão especificada por `id`;

(3) `status` – apresentar o estado atual das variáveis contidas no `data_container`;

(4) `help` – mostrar as informações de ajuda sobre as opções anteriores.

(5) `end` – terminar a execução do sistema *hOSpital*;

Quando se pretende dar entrada no hospital (1), o utilizador pede à `Main` (através de uma interface linha de comandos) para criar uma senha de admissão, especificando o seu identificador de paciente e o do médico pretendido. A senha de admissão é primeiro enviada para o paciente em questão, que adiciona o seu identificador como confirmação de recebimento da mesma. De seguida o paciente envia esta senha de admissão para os rececionistas, onde qualquer rececionista disponível pode fazer a sua admissão. O rececionista que processar o pedido acrescenta o seu identificador à admissão e envia a mesma para o médico pretendido. Quando disponível, o médico acrescenta o seu identificador como confirmação de recebimento da admissão, e modifica o estado para (A) de atendimento ao paciente ou (N) para indicar a necessidade de uma nova marcação, dependendo se ainda tem capacidade para realizar o atendimento no próprio dia (e.g., nesta execução do *hOSpital*).

O modelo de interação entre os 3 participantes é o modelo produtor/consumidor. Um atendimento em curso transita entre cinco estados que indicam em que estado o seu processamento se encontra. Os estados são: '`M`' – senha de admissão iniciada pela `Main`; '`P`' – senha de admissão validada pelo paciente; '`R`' – admissão realizada por um rececionista; '`A`' – atendimento realizado pelo médico; ou '`N`' – caso uma nova consulta tenha que ser agendada. Os estados '`A`' e '`N`' são mutuamente exclusivos: o médico regista o atendimento num estado ou no outro, dependendo de o número máximo de consultas do dia (i.e., da execução do *hOSpital*) ter sido ultrapassado, mas nunca nos dois.

A ação de consulta de estado de uma senha de admissão (2), por sua vez, verifica e imprime o estado da admissão. Esta impressão incluirá o identificador do paciente, o identificador do médico pretendido, o identificador do paciente que efetivamente validou a senha de admissão, o identificador do rececionista que realizou a admissão, o identificador do médico que efetivamente confirmou a receção da admissão (e consequentemente atendeu o paciente ou indicou que é necessária uma nova marcação). A ação de consulta `status` (3) vai imprimir o estado atual das variáveis do `data_container`. A ação `help` (4) vai apresentar o menu com as opções de ações disponíveis.

Por fim, a ação para terminar a execução do sistema *hOSpital* (5) imprime as estatísticas finais do mesmo, como o número de admissões pedidas por cada paciente, o número de admissões realizadas por cada rececionista e o número de atendimentos de cada médico, e termina o programa.

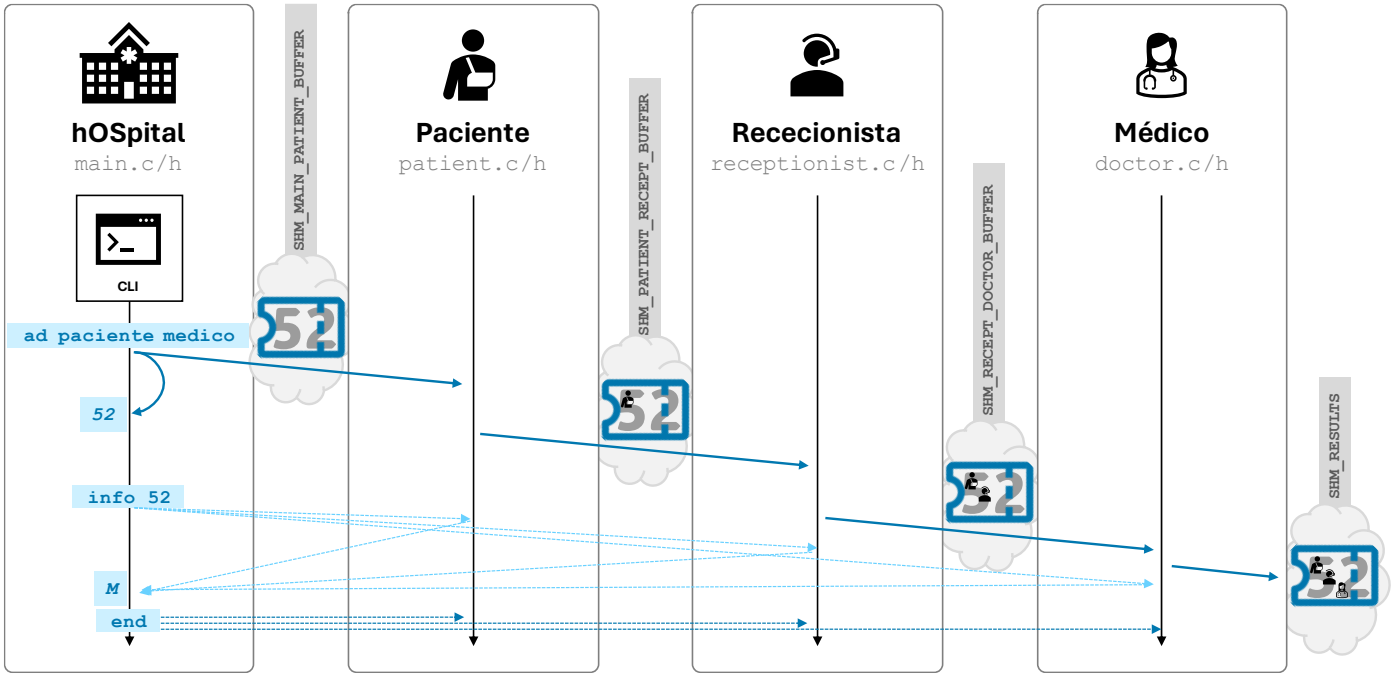

Figura 1: Visão geral do sistema ***hOSpital*** e as interações entre os participantes.

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 realiza a interação com o utilizador.
- Gestão de processos (`process.c/h`), com funções para criar e destruir processos (e.g., Pacientes, Rececionistas e Médicos).
- 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 (e.g., buffers circulares e de acesso aleatório que serão apresentados na TP05).
- Paciente (`patient.c/h`), com funções para receber senhas de admissão da `Main`, atribuir o seu identificador de paciente e encaminhá-las para os Rececionistas.
- Rececionista (`receptionist.c/h`), com funções para receber pedidos de admissão de Pacientes e encaminhar os mesmos para os Médicos.
- Médico (`doctor.c/h`), com funções para receber os pedidos de atendimento dos rececionistas, atender os pacientes e enviar para a `Main` estatísticas dos mesmos.

Para cada um destes módulos, é fornecido pelos professores 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 respetivos ficheiros `X.c`, utilizando os algoritmos e métodos que o grupo achar convenientes. Se o grupo entender necessário, ou se for pedido, pode-se criar um ficheiro de cabeçalho `X-private.h` para acrescentar outras definições, cujas implementações serão também incluídas nos ficheiros `X.c`.

Juntamente com o enunciado da 1ª Fase do projeto é disponibilizado, aos alunos, um ficheiro ZIP contendo todos os cabeçalhos definidos para os módulos acima apresentados.

Para auxiliar no início do desenvolvimento do projeto, é também fornecida uma função `main` que os alunos podem usar como base:

```
int main(int argc, char *argv[]) {
	//init data structures
	struct data_container* data = allocate_dynamic_memory(sizeof(struct data_container));
	struct communication* comm = allocate_dynamic_memory(sizeof(struct communication));
	comm->main_patient = allocate_dynamic_memory(sizeof(struct circular_buffer));
	comm->patient_receptionist = allocate_dynamic_memory(sizeof(struct rnd_access_buffer));
	comm->receptionist_doctor = allocate_dynamic_memory(sizeof(struct circular_buffer));

	//execute main code
	main_args(argc, argv, data);
	allocate_dynamic_memory_buffers(data);
	create_shared_memory_buffers(data, comm);
	launch_processes(data, comm);
	user_interaction(data, comm);

	//release memory before terminating
	deallocate_dynamic_memory(data);
	deallocate_dynamic_memory(comm->main_patient);
	deallocate_dynamic_memory(comm->patient_receptionist);
	deallocate_dynamic_memory(comm->receptionist_doctor);
	deallocate_dynamic_memory(comm);
}
```

4 Estrutura do Projeto e makefile

O projeto deve ser organizado segundo a seguinte estrutura:

```
\HOSPITAL
	\bin
	\include
	\obj
	\src
```

O diretório `bin` deverá conter o executável *hOSpital*. 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 os códigos desenvolvidos pelos alunos.

O executável *hOSpital* 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 HOSPITAL.

5 Desenvolvimento e Testes

Para efeitos de teste, é fornecido aos alunos o executável `hOSpital_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 `hOSpital` (e do `hOSpital_profs`):

```
$./hOSpital max_consultas buffers_size n_patients n_rececionists n_doctors
```

, onde `max_consultas` é o número máximo de consultas que podem ser admitidas durante a execução do programa (e.g., a representar um dia do hospital), `buffers_size` é o tamanho máximo dos *buffers* (i.e., o número máximo de elementos que podem estar ao mesmo tempo no *buffer*) em memória partilhada, `n_patients` é o número de pacientes, `n_receptionists` é o número de rececionistas e `n_doctors` é o número de Médicos.

Recomenda-se aos alunos que comecem os testes com um número reduzido de operações e processos (e.g., 10 admissões, 1 paciente, 1 rececionista e 1 médico). Caso o código funcione com este exemplo, então aumentem e testem com mais processos e admissões.

De notar que nesta 1ª fase do projeto, e no caso de haver vários rececionistas, pode acontecer que mais que um receba e tente realizar a mesma admissão. Este problema demonstra a necessidade de efetuar a sincronização entre os processos, o que será resolvido na 2ª fase do projeto.

Depois do arranque da aplicação, o utilizar deve interagir com o sistema por meio de um menu com as opções `ad`, `info`, `status` e `end` que foram explicadas na Secção 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 Secção 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 `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 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/.

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

7 Prazo de Entrega

O trabalho deve ser entregue até dia **14 de abril de 2024** (domingo) **às 23:59h**. Após esta data, a submissão do trabalho através do Moodle deixará de ser permitida.

8 Avaliação dos Trabalhos

A avaliação do trabalho será realizada:

(1) pelos alunos, pelo preenchimento do 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 avaliados: funcionalidade, estrutura, desempenho, algoritmia, comentários, clareza do código, validação dos parâmetros de entrada e tratamento de erros.

9 Divulgação dos Resultados

A data prevista da divulgação dos resultados da avaliação dos trabalhos é 12 de maio de 2024.

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.