Projeto de Sistemas Operativos 2023-24
Enunciado da 2ª parte do projeto LEIC-A/LEIC-T/LETI

A segunda parte do projeto consiste de 2 exercícios que visam:

i) tornar o IST-EMS acessível a processos clientes através de *named pipes*,
ii) permitir a interação com o IST-EMS através de sinais.

Código base

O código base fornecido disponibiliza uma implementação do servidor que corresponde a uma possível solução da primeira parte do projeto, sem todo o código relacionado com leitura de ficheiros (foi movido para o cliente) e de criação de threads e processos (contém especialmente a lógica de sincronização entre tarefas, maioritariamente no ficheiro operations.c). Também contém uma implementação da API de cliente vazia, e um cliente que recebe o caminho para um ficheiro *.jobs*, que chama a API para cada um dos comandos no ficheiro.

Os comandos utilizados nesta parte do projeto são os mesmos da primeira parte, com exceção do **BARRIER**, que já não existe nesta entrega, e do **WAIT**, que já não recebe o argumento *thread_id*, e é executado do lado do cliente.

Exercício 1. Interação com processos clientes por *named pipes*

O IST-EMS deve passar a ser um processo servidor autónomo, lançado da seguinte forma:

```
ems nome_do_pipe
```

Quando se inicia, o servidor deve criar um *named pipe* cujo nome (*pathname*) é o indicado no argumento acima. É através deste *pipe* que os processos clientes se poderão ligar ao servidor e enviar pedidos de início de sessão.

Qualquer processo cliente pode ligar-se ao *pipe* do servidor e enviar-lhe uma mensagem a solicitar o início de uma sessão. Esse pedido contém os nomes de dois *named pipe*, que o cliente previamente criou para a nova sessão. É através destes *named pipes* que o cliente enviará futuros pedidos para o servidor e receberá as correspondentes respostas do servidor no âmbito da nova sessão.

Ao receber um pedido de sessão, o servidor atribui um identificador único à sessão, designado *session_id*, e associa a esse *session_id* os nomes dos *named pipes* que o cliente indicou. De seguida, responde ao cliente com o *session_id* da nova sessão.

O servidor aceita no máximo S sessões em simultâneo, cada uma com um *session_id* distinto, sendo que *session_id* é um valor entre [0, S - 1], em que S é uma constante definida no código do servidor. Isto implica que o servidor, quando recebe um novo pedido de início de sessão e tem S sessões ativas, deve bloquear, esperando que uma sessão termine para que possa criar a nova.

Uma sessão dura até ao momento em que i) o cliente envia uma mensagem de fim de sessão ou que ii) o servidor detecta que o cliente está indisponível. Nas subsecções seguintes descrevemos a API cliente do IST-EMS em maior detalhe, assim como o conteúdo das mensagens de pedido e resposta trocadas entre clientes e servidor.

API cliente do IST-EMS

Para permitir que os processos cliente possam interagir com o IST-EMS, existe uma interface de programação (API), em C, a qual designamos por API cliente do IST-EMS. Esta API permite ao cliente ter programas que estabelecem uma sessão com um servidor e, durante essa sessão, invocar operações para aceder e modificar o estado dos eventos geridos pelo IST-EMS. De seguida apresentamos essa API.

As seguintes operações permitem que o cliente estabeleça e termine uma sessão com o servidor:

- int ems_setup (char const *req_pipe_path, char const *resp_pipe_path, char const *server_pipe_path)
  Estabelece uma sessão usando os *named pipes* indicados em argumento.
  Os *named pipes* usados pela troca de pedidos e respostas (isto é, após o estabelecimento da sessão) devem ser criados (chamando *mkfifo)* usando os nomes passados no 1º e 2º argumento. O *named pipe* do servidor deve já estar previamente criado pelo servidor, e o correspondente nome é passado no 3º argumento.
  Em caso de sucesso, o *session_id* associado à nova sessão terá sido guardado numa variável do cliente que indica qual a sessão que o cliente tem ativa neste momento; adicionalmente, todos os *pipes* terão sido abertos pelo cliente.
  Retorna 0 em caso de sucesso, 1 em caso de erro.

- int ems_quit()
  Termina uma sessão ativa, identificada na variável respectiva do cliente, fechando os *named pipes* que o cliente tinha aberto quando a sessão foi estabelecida e apagando o *named pipe* cliente.
  Retorna 0 em caso de sucesso, 1 em caso de erro.

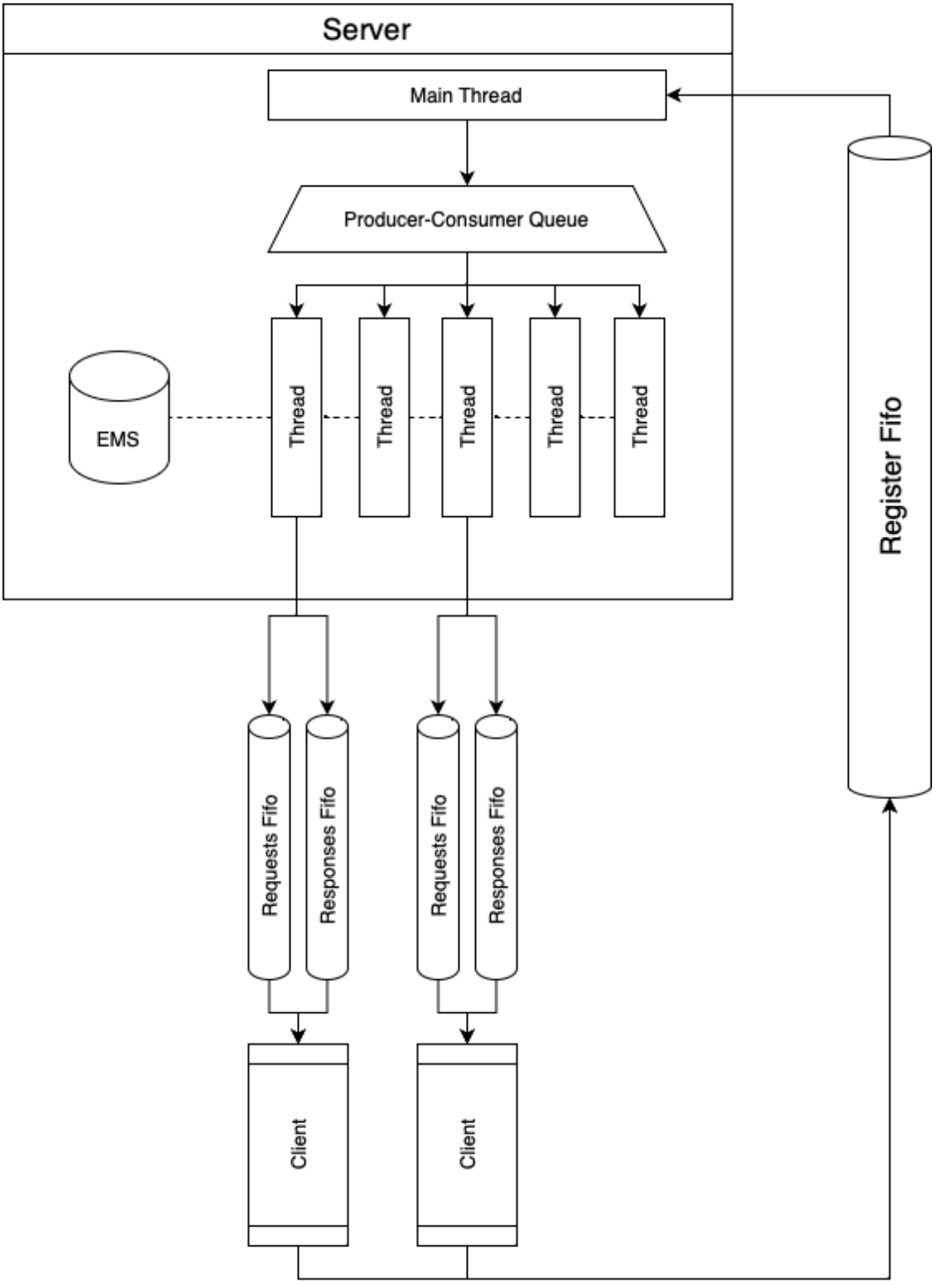
Tendo uma sessão ativa, o cliente pode invocar as seguintes operações junto do servidor, cuja especificação é idêntica à das operações homónimas do servidor:

- int ems_create(unsigned int event_id, size_t num_rows, size_t num_cols)
- int ems_reserve(unsigned int event_id, size_t num_seats, size_t* xs, size_t* ys)
- int ems_show(int out_fd, unsigned int event_id)
- int ems_list_events(int out_fd)

Tanto o ems_show como o ems_list_events recebem um *file descriptor* para onde devem imprimir o seu output, com o mesmo formato da primeira parte do projeto.

Diferentes programas cliente podem existir, todos eles invocando a API acima indicada (concorrentemente entre si). Por simplificação, devem ser assumidos estes pressupostos:

- Os processos cliente são *single-threaded*, ou seja a interação de um cliente com o servidor é sequencial (um cliente só envia um pedido depois de ter recebido a resposta ao pedido anterior).
- Os processos cliente são corretos, ou seja cumprem a especificação que é descrita no resto deste documento. Em particular, assume-se que nenhum cliente envia mensagens com formato fora do especificado.



Protocolo de pedidos-respostas

O conteúdo de cada mensagem (de pedido e resposta) deve seguir o seguinte formato:

Função da API cliente
int ems_setup(char const *req_pipe_path, char const* resp_pipe_path, char const *server_pipe_path)
Mensagens de pedido e resposta
(char) OP_CODE=1 | (char[40]) nome do pipe do cliente (para pedidos) | (char[40]) nome do pipe do cliente (para respostas)
(int) session_id

Função da API cliente
int ems_quit(void)
Mensagens de pedido e resposta
(char) OP_CODE=2
<sem resposta>

Função da API cliente
int ems_create(unsigned int event_id, size_t num_rows, size_t num_cols)
Mensagens de pedido e resposta
(char) OP_CODE=3 | (unsigned int) event_id | (size_t) num_rows | (size_t) num_cols
(int) retorno (conforme código base)

Função da API cliente
int ems_reserve(unsigned int event_id, size_t num_seats, size_t* xs, size_t* ys)
Mensagens de pedido e resposta
(char) OP_CODE=4 | (unsigned int) event_id | (size_t) num_seats | (size_t[num_seats]) conteúdo de xs | (size_t[num_seats]) conteúdo de ys
(int) retorno (conforme código base)

Função da API cliente
int ems_show (int out_fd, unsigned int event_id)
Mensagens de pedido e resposta
(char) OP_CODE=5 | (unsigned int) event_id
(int) retorno (conforme código base) | (size_t) num_rows | (size_t) num_cols | (unsigned int[num_rows * num_cols]) seats

Função da API cliente

int ems_list_events (int out_fd)
Mensagens de pedido e resposta
(char) OP_CODE=6
(int) retorno (conforme código base) | (size_t) num_events | (unsigned int[num_events]) ids

Onde:

- O símbolo | denota a concatenação de elementos numa mensagem. Por exemplo, a mensagem de pedido associada à função ems_quit consiste num byte (char) seguido de um inteiro (int).
- Todas as mensagens de pedido são iniciadas por um código que identifica a operação solicitada (*OP_CODE*). Com a exceção dos pedidos de *ems_setup*, o *OP_CODE* é seguido do *session_id* da sessão atual do cliente (que deverá ter sido guardado numa variável do cliente aquando da chamada a *ems_setup*).
- As *strings* que transportam os nomes de *pipes* são de tamanho fixo (40). No caso de nomes de tamanho inferior, os caracteres adicionais devem ser preenchidos com '\0'.
- O buffer de lugares devolvido pelo ems_show deve seguir a ordem principal de linha (*row-major order*).
- Em caso de erro no ems_show ou no ems_list_events, o servidor deve enviar apenas o código de erro.

Implementação em duas etapas

Dada a complexidade deste requisito, recomenda-se que a solução seja desenvolvida de forma gradual, em 2 etapas que descrevemos de seguida.

Etapa 1.1: Servidor IST-EMS com sessão única

Nesta fase, devem ser assumidas as seguintes simplificações (que serão eliminadas no próximo requisito):

- O servidor é *single-threaded*.
- O servidor só aceita uma sessão de cada vez (ou seja, S=1).

Experimente:

Corra o teste disponibilizado em jobs*/test.jobs* sobre a sua implementação cliente-servidor do IST-EMS. Confirme que o teste termina com sucesso.

Construa e experimente testes mais elaborados que exploram diferentes funcionalidades oferecidas pelo servidor IST-EMS.

Etapa 1.2: Suporte a múltiplas sessões concorrentes

Nesta etapa, a solução composta até ao momento deve ser estendida para suportar os seguintes aspetos mais avançados.

Por um lado, o servidor deve passar a suportar múltiplas sessões ativas em simultâneo (ou seja, S>1).
Por outro lado, o servidor deve ser capaz de tratar pedidos de sessões distintas (ou seja, de clientes distintos) em paralelo, usando múltiplas tarefas (*pthreads)*, entre as quais:

- A tarefa inicial do servidor deve ficar responsável por receber os pedidos que chegam ao servidor através do seu *pipe*, sendo por isso chamada a *tarefa anfitrião*.
- Existem também S tarefas trabalhadoras, cada uma associada a um *session_id* e dedicada a servir os pedidos do cliente correspondente à esta sessão. As tarefas trabalhadoras devem ser criadas aquando da inicialização do servidor.

A tarefa anfitrião coordena-se com as tarefas trabalhadoras da seguinte forma:

- Quando a tarefa anfitrião recebe um pedido de estabelecimento de sessão por um cliente, a tarefa anfitrião insere o pedido num buffer produtor-consumidor. As tarefas trabalhadoras extraem pedidos deste *buffer* e comunicam com o respectivo cliente através dos named pipes que o cliente terá previamente criado e comunicado junto ao pedido de estabelecimento da sessão. A sincronização do *buffer* produtor-consumidor deve basear-se em variáveis de condição (além de *mutexes*).

Experimente:

Experimente correr os testes cliente-servidor que compôs anteriormente, mas agora lançando-os concorrentemente por 2 ou mais processos cliente.

Exercício 2. Interação por sinais

Estender o IST-EMS de forma que no servidor seja redefinida a rotina de tratamento do SIGUSR1. Ao receber este sinal o (servidor) IST-EMS deve memorizar que, o mais breve possível, mas fora da função de tratamento do sinal, a thread principal deverá imprimir no std-output o identificador de cada evento, seguido do estado dos seus lugares, tal como o SHOW do primeiro exercício. Dado que só a *thread* principal que recebe as ligações dos clientes deve escutar o SIGUSR1, todas as *threads* de atendimento de um cliente específico devem usar a função pthread_sigmask para inibirem (com a opção SIG_BLOCK) a recepção do SIGUSR1.

Ponto de partida.

Para resolver a 2ª parte do projeto, os grupos podem optar por usar como base a sua solução da 1ª parte do projeto ou aceder ao novo código base. Caso se opte por usar a solução da 1ª parte do projeto como ponto de partida, poder-se-á aproveitar a lógica de sincronização entre tarefas. Contudo, nesta fase do projeto o servidor do IST-EMS corre apenas num processo. Portanto as extensões desenvolvidas na primeira fase do projeto

para obter paralelização por múltiplos processos não poderão ser aproveitadas para esta parte do projeto.

Submissão e avaliação

A submissão é feita através do Fénix **até ao dia 5 de Janeiro às 23h59**.

Os alunos devem submeter um ficheiro no formato *zip* com o código fonte e o ficheiro *Makefile*. O arquivo submetido não deve incluir outros ficheiros (tais como binários). Além disso, o comando *make clean* deve limpar todos os ficheiros resultantes da compilação do projeto.

Recomendamos que os alunos se assegurem que o projeto compila/corre corretamente no cluster *sigma*. Ao avaliar os projetos submetidos, em caso de dúvida sobre o funcionamento do código submetido, os docentes usarão o cluster sigma para fazer a validação final.

O uso de outros ambientes para o desenvolvimento/teste do projeto (e.g., macOS, Windows/WSL) é permitido, mas o corpo docente não dará apoio técnico a dúvidas relacionadas especificamente com esses ambientes.

A avaliação será feita de acordo com o método de avaliação descrito no site da cadeira.