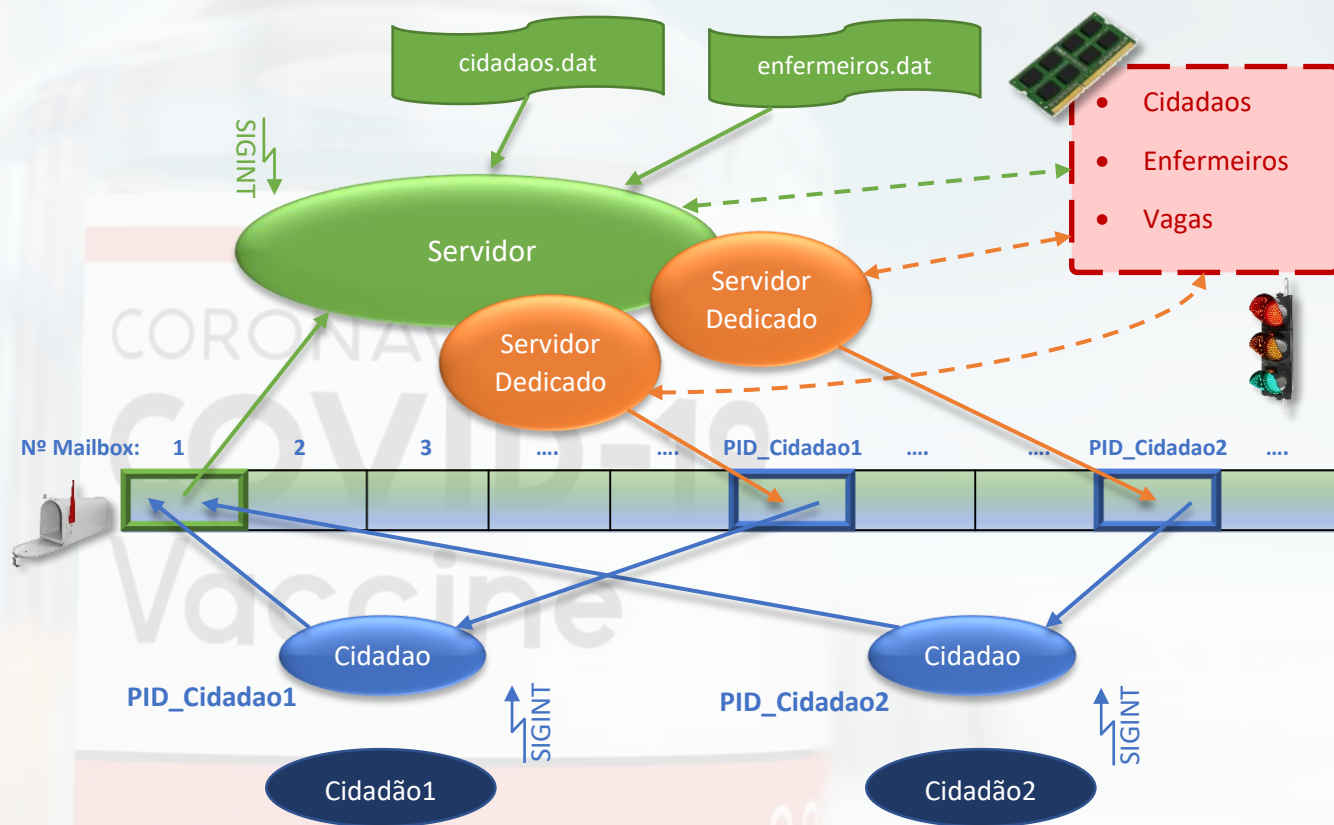


Projeto Covid-IUL (Parte 3)

Nesta parte do trabalho, será implementado um modelo simplificado da Administração de vacinas dos cidadãos do sistema Covid-IUL, baseado em comunicação por IPC entre processos, utilizando a linguagem de programação C. Considere o seguinte diagrama, que apresenta uma visão geral da arquitetura pretendida:



Ideia Geral: Pretende-se, nesta fase, simular a realização da vacinação no sistema Covid-IUL. Assim, teremos dois módulos a realizar – **Cidadão** e **Servidor**.

Para distinguir os destinatários das mensagens, vamos utilizar o campo **tipo** (Address) da **mensagem**:

- Se **tipo = 1**, a mensagem é destinada ao servidor
- Se **tipo = PID_Cidadao** (obrigatoriamente maior que 1), a mensagem é destinada ao processo cujo *process ID* é igual a *PID_Cidadao*.

Entrega, relatório e autoavaliação

O trabalho de SO será realizado **individualmente**, logo sem recurso a grupos.

A entrega da Parte 3 do trabalho será realizada através da criação de **um** ficheiro ZIP cujo nome é o nº do aluno, e.g., “**a<nºaluno>-parte-3.zip**” (**ATENÇÃO: não serão aceites ficheiros RAR, 7Z ou outro formato**) onde estarão todos os ficheiros criados. Estes serão apenas os ficheiros de código, ou seja, na terceira parte, apenas os ficheiros de código (.c e .h). Cada um dos módulos será desenvolvido com base nos ficheiros fornecidos, e que estão na diretoria do Tigre “/home/so/trabalho-2020-2021/parte-3”, e deverá incluir nos comentários iniciais um “relatório” indicando a descrição do módulo e explicação do mesmo (poderá ser muito reduzida se o código tiver comentários bem descritivos). Naturalmente, deverão copiar todos estes ficheiros para a vossa área.

Para criarem o ficheiro ZIP, deverão usar, no Tigre, o comando **\$ zip a<nº aluno>-parte-3.zip <ficheiros>**, por exemplo:

```
$ zip a123456-parte-3.zip *.c *.h
```

A entrega desta parte do trabalho deverá ser feita por via eletrónica, através do e-learning:

- e-learning da UC Sistemas Operativos;
- Seleccionam a opção sub-menu “Conteúdo/Content”;
- Seleccionem o link “Trabalho Prático 2020/2021 Parte 3”;
- Dentro do formulário “Visualizar Exercício de carregamento: Trabalho Prático 2020/2021 Parte 3”, seleccionem “Anexar Arquivo” e anexem o vosso ficheiro .zip. Podem submeter o vosso trabalho as vezes que desejarem, **apenas a última submissão será contabilizada**. Certifiquem-se que a submissão foi concluída, e que esta última versão tem todas as alterações que desejam entregar dado que os docentes apenas considerarão esta última submissão;
- Avisamos que a hora de entrega final acontece sempre poucos **minutos antes da meia-noite**, pelo que se urge a que os alunos não esperem pela hora final para entregarem e o façam, idealmente um dia antes, ou no pior caso, pelo menos uma hora antes. **Não serão consideradas válidas as entregas realizadas por e-mail**. Poderão testar a entrega nos dias anteriores para perceberem se têm algum problema com a entrega, sendo que, novamente, apenas a última submissão conta.

Política em caso de fraude

O trabalho entregue deve corresponder ao esforço individual de cada aluno. São consideradas fraudes as seguintes situações: Trabalho parcialmente copiado, facilitar a cópia através da partilha de ficheiros, ou utilizar material alheio sem referir a sua fonte.

Em caso de deteção de algum tipo de fraude, os trabalhos em questão não serão avaliados, sendo enviados à Comissão Pedagógica ou ao Conselho Pedagógico, consoante a gravidade da situação, que decidirão a sanção a aplicar aos alunos envolvidos. Serão utilizadas as ferramentas *Moss* e *SafeAssign* para deteção automática de cópias. Recorda-se ainda que o Anexo I do Código de Conduta Académica, publicado a 25 de janeiro de 2016 em Diário da República, 2ª Série, nº 16, indica no seu ponto 2 que quando um trabalho ou outro elemento de avaliação apresentar um nível de coincidência elevado com outros trabalhos (percentagem de coincidência com outras fontes reportada no relatório que o referido software produz), cabe ao docente da UC, orientador ou a qualquer elemento do júri, após a análise qualitativa desse relatório, e em caso de se confirmar a suspeita de plágio, desencadear o respetivo procedimento disciplinar, de acordo com o Regulamento Disciplinar de Discentes do ISCTE - Instituto Universitário de Lisboa, aprovado pela deliberação nº 2246/2010, de 6 de dezembro.

O ponto 2.1 desse mesmo anexo indica ainda que no âmbito do Regulamento Disciplinar de Discentes do ISCTE-IUL, são definidas as sanções disciplinares aplicáveis e os seus efeitos, podendo estas variar entre a advertência e a interdição da frequência de atividades escolares no ISCTE-IUL até cinco anos.

Parte III – Comunicação usando IPC

Data de entrega: **16 de maio de 2021**

Nesta parte do trabalho, os ficheiros a utilizar (os nomes devem ser exatamente estes, usando só minúsculas) são:

- “**cidadaos.dat**” e “**enfermeiros.dat**”: Ficheiros binários com a informação sobre os Cidadãos e Enfermeiros, fornecido pelo sistema (copiar para a vossa diretoria a partir da diretoria existente no servidor Tigre `/home/so/trabalho-2020-2021/parte-3/`). **Os alunos deverão assumir que estes ficheiros podem vir a ter outros conteúdos ou outros tamanhos.** A estrutura dos ficheiros é, respetivamente, **Cidadao** e **Enfermeiro**.

As estruturas de dados necessárias para elaborar os módulos desta parte são **Cidadao**, **Enfermeiro** e **Vaga**:

(ficheiros disponibilizados aos alunos no Tigre acima)

cidadao.c: Módulo processo Cidadão

servidor.c: Módulo processo Servidor

common.h: Definições comuns a Cidadão e Servidor

utils.h: Macros utilitárias

cidadaos.dat: Ficheiro binário com base de dados de cidadãos, pode ser lido usando o Shell script **show-cidadaos.sh**

enfermeiros.dat: Ficheiro binário com lista de enfermeiros, pode ser lido usando o Shell script **show-enfermeiros.sh**

```
typedef struct {
    int num_utente;
    char nome[100];
    int idade;
    char localidade[100];
    char nr_telemovei[10];
    int estado_vacinacao;
    int PID_cidadao;
} Cidadao;

typedef struct {
    int ced_profissional;
    char nome[100];
    char CS_enfermeiro[100];
    int nr_vacinas_dadas;
    int disponibilidade;
} Enfermeiro;

typedef struct {
    int index_enfermeiro;
    int index_cidadao;
    int PID_filho;
} Vaga;
```

```
typedef struct {
    long tipo;
    struct {
        TipoPedido pedido;
        int num_utente;
        char nome[100];
        int PID_cidadao;
    } dados;
} MsgCliente;

typedef struct {
    long tipo;
    struct {
        StatusServidor status;
        Cidadao cidadao;
    } dados;
} MsgServidor;

typedef struct {
    Cidadao cidadaos[MAX_CIDADAOS];
    int num_cidadaos;
    Enfermeiro enfermeiros[MAX_ENFERMEIROS];
    int num_enfermeiros;
    Vaga vagas[MAX_VAGAS];
} Database;
```


Desafios de negócio do projeto:

Sendo este um processo que está sujeito a erros e falhas de negócio, passam-se a enunciar as formas que foram encontradas de evitar as mesmas, exemplificando vários casos de uso:

- Cidadão não existe na BD de cidadãos

- Nesse caso, o servidor não encontra o cidadão na BD, e retorna uma resposta indicando status= DESCONHECIDO;
- O cliente (cidadão) quando receber essa resposta tem de sair sem fazer vacinação.

- Cidadão é de uma localidade que não existe ou que não tem CS ou que não tem enfermeiro

- Nesse caso, o servidor avalia que não existe nenhum enfermeiro para o CS da localidade do cidadão (para efeitos de se poder testar esta funcionalidade, alguns cidadãos existentes na BD moram na localidade "LocalidadeSemEnfermeiro";
- Nesse caso, o servidor tem de retornar uma resposta com status=NAOHAENFERMEIRO;
- O cliente (cidadão) quando receber essa resposta tem de sair sem fazer vacinação.

- Dois ou mais cidadãos da mesma localidade foram ser vacinados ao mesmo tempo

- Quando o servidor proceder à vacinação, tem de colocar na BD de Enfermeiros o enfermeiro em questão como indisponível (disponibilidade=0) para futuras vacinas;
- Se, no decorrer da sessão de vacinação anterior, outro cidadão chegar que seja da mesma localidade (logo, para o mesmo Enfermeiro), então o servidor deteta que está indisponível e retorna status= AGUARDAR;
- Nesta situação, o cliente (cidadão) irá aguardar um tempo TEMPO_ESPERA segundos e depois tentará novamente enviar novo pedido de vacinação;
- Não esquecer de que quando o Enfermeiro terminar a sessão de vacinação, o servidor tem de voltar a colocar o Enfermeiro como disponível (disponibilidade=1).

- O mesmo cidadão tentou ser vacinado novamente enquanto ainda está numa sessão de vacina

- Esta situação é diferente da anterior, porque é certamente fruto de um erro, logo **não queremos** que, o cliente (cidadão) aguarde um tempo TEMPO_ESPERA segundos e depois tente novamente; é antes, suposto que o cidadão termine a sua aplicação;
- Quando o servidor receber o pedido de vacinação, vai à procura dos dados do cidadão na BD Cidadaos, e se encontrar, altera o campo PID_cidadao (que normalmente tem o valor de -1) para ficar com o PID do processo cliente;
- Se, no decorrer da sessão de vacinação anterior, o mesmo cidadão voltar a tentar ser vacinado, quando o servidor encontrar o Cidadão na BD Clientes, vai verificar que PID_cidadao > 0 e retorna status= EMCURSO;
- O cliente (cidadão) quando receber essa resposta tem de sair sem fazer vacinação;
- Não esquecer de que quando terminar a sessão de vacinação, o servidor tem de ir novamente ao registo do utilizador que acabou de ser vacinado e dar check-out do cidadão (coloca PID_cidadao=-1).

- **Cidadão já tomou as 2 doses da vacina e voltou para mais**

- Cada cidadão está associado a um campo estado_vacinacao, que se inicia sempre a 0;
- Quando o servidor termina a vacinação de um cidadão, incrementa este campo;
- Quando o servidor recebe um pedido de vacinação, vai à procura dos dados do cidadão na BD Cidadãos, e se encontrar, valida se o estado_vacinacao já chegou a 2. Se for esse o caso, não prossegue a vacinação e retorna status=VACINADO;
- O cliente (cidadão) quando receber essa resposta tem de sair sem fazer vacinação;

- **Cidadãos de localidades diferentes foram ser vacinados ao mesmo tempo e encheram as vagas**

- Quando o servidor proceder à vacinação, vai analisar se existe vaga para a sessão de vacinação. Se não houver vaga, então o servidor não prossegue com o pedido e retorna status=AGUARDAR;
- Nesta situação, o cliente (cidadão) irá aguardar um tempo TEMPO_ESPERA segundos e depois tentará novamente enviar novo pedido de vacinação.

- **Cidadão que estava prestes a ser vacinado não se sentiu bem e desistiu**

- Nesta situação, é possível ao cliente (Cidadao) pressionar <CTRL-C> para desistir;
- Com esta operação, o cliente (Cidadao) deverá enviar um pedido de cancelamento de sessão ao servidor;
- Este pedido está sujeito às mesmas prioridades que todos os outros, pelo que poderá resultar num cancelamento ou não:
 - O servidor contacta o Servidor Dedicado que está a tomar conta da sessão e dá a indicação que a consulta deverá ser cancelada. Se tudo correr bem, o Servidor Dedicado envia ele próprio uma resposta ao Cliente a confirmar que a sessão foi cancelada (com status=CANCELADA);
 - No entanto, pode acontecer que o pedido de cancelamento só tenha dado entrada quando o Cidadão já estava a ser vacinado ou já o tinha sido. Nessa situação, o servidor envia uma resposta com status=TERMINADA, indicando que apesar de ter sido pedido o cancelamento, a vacinação prosseguiu e terminou.

- **Servidor precisa parar para manutenção**

- Nesta situação, é possível ao servidor pressionar <CTRL-C> para se desligar;
- Com esta operação, o servidor deverá contactar todos os Servidores Dedicados ativos que estão a tomar conta das sessões de vacinação, e dá a indicação que as suas consultas deverão ser canceladas. Isto resultará que cada Servidor Dedicado envia ele próprio uma resposta ao Cliente a indicar que a sessão foi cancelada (com status=CANCELADA).

- **Situação normal e standard (finalmente)**

- O Cidadão envia uma mensagem de pedido de vacinação ao servidor;
- O servidor faz todas as validações e verifica que estão todas OK;
- O servidor cria um servidor dedicado para tomar conta da sessão de vacinação;
- O servidor dedicado envia uma resposta ao Cidadão com status=OK (início da vacinação);
- O servidor dedicado envia uma resposta ao Cidadão com status=TERMINADA (fim do processo de vacinação);

Os alunos deverão, em vez de `printf`, utilizar sempre as macros “sucesso” e “erro” (definidas em `utils.h`; a sintaxe destas macros é igual à do `printf`) para escrever TODAS as mensagens, respetivamente, de sucesso e erro resultantes dos vários passos da aplicação.

cidadao.c

O módulo **Cidadão** simula, na prática, a chegada do cidadão ao centro de saúde da sua localidade para iniciar o processo de vacinação, seguindo as regras do plano de Vacinação.

Assim, definem-se as seguintes tarefas a desenvolver:

- C1)** Implemente a função `init_ipc()`, que tenta abrir uma fila de mensagens IPC que tem a KEY `IPC_KEY` definida em `common.h` (alterar esta KEY para ter o valor do nº do aluno, como indicado nas aulas). Deve assumir que a fila de mensagens já foi criada. Se tal não aconteceu, dá erro e termina com *exit status 1*. Esta função, em caso de sucesso, preenche a variável global `msg_id`;

Outputs esperados (os itens entre <> deverão ser substituídos pelos valores correspondentes):

@@Error@@: C1) Fila de Mensagens com a Key definida não existe ou não pode ser aberta

@@Success@@: C1) Fila de Mensagens com a Key <IPC_KEY> aberta com o ID <msg_id>

- C2)** Implemente a função `cria_mensagem()`, que:

- C2.1)** Pede ao **Cidadão** (utilizador) os seus dados, nomeadamente o número de utente e nome, obrigatoriamente nessa ordem, preenchendo os dados na variável global `mensagem`;

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

@@Success@@: C2.1) Dados Cidadão: <num_utente>, <nome>

- C2.2)** Preenche os campos `PID_cidadao` da variável global `mensagem` com o PID deste **processo Cidadão**, tipo da mensagem com o **tipo 1**, e **pedido = PEDIDO**;

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

@@Success@@: C2.2) PID Cidadão: <PID_Cidadao>

- C3)** Implemente a função `envia_mensagem_servidor()` que envia um pedido de consulta de vacinação para o **processo Servidor**, enviando a `mensagem` para a fila de mensagens; em caso de erro no envio, termina com erro e *exit status 1*;

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

@@Error@@: Não é possível enviar mensagem para o servidor

@@Success@@: Mensagem para o servidor enviada

- C4)** Implemente a função `espera_resposta_servidor()`, que espera a resposta do **processo Servidor** (na fila de mensagens com o **tipo = PID_Cidadao**) e preenche a mensagem enviada pelo **processo Servidor** na variável global `resposta`; em caso de erro, termina com erro e *exit status 1*.

Outputs esperados (os itens entre <> deverão ser substituídos pelos valores correspondentes):

@@Error@@: Não é possível ler a resposta do servidor

@@Success@@: Servidor enviou resposta

C5) O comportamento do **processo Cidadão** agora irá depender da **resposta** enviada pelo **processo Servidor** no campo **status**. Implemente a função **trata_resposta_servidor()** que faz:

C5.1) Se o estado for **DESCONHECIDO** ou **NAOHAENFERMEIRO**, imprime uma mensagem de erro, e termina o processo com *exit status 1*;

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

@@Error@@: C5.1) Não existe registo do utente <num_utente>, <nome>

@@Error@@: C5.1) Não existe enfermeiro na localidade do utente <num_utente>, <nome>

C5.2) Se o estado for **VACINADO** ou **EMCURSO**, imprime uma mensagem de sucesso, e termina o processo com *exit status 0*;

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

@@Success@@: C5.2) O utente <num_utente>, <nome> já foi vacinado

@@Success@@: C5.2) A vacinação do utente <num_utente>, <nome> já está em curso

C5.3) Se o estado for **AGUARDAR**, imprime uma mensagem de sucesso, aguarda (sem espera ativa!) um tempo **TEMPO_ESPERA** segundos, e depois retorna ao ponto **C3**, enviando novo pedido;

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

@@Success@@: C5.4) Utente <num_utente>, <nome>, por favor aguarde...

C5.4) Se o estado for **OK**, imprime uma mensagem de sucesso, e depois vai para o ponto **C6**.

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

@@Success@@: C5.5) Utente <num_utente>, <nome>, vai agora ser vacinado

C6) Inicia o processo de vacinação. Implemente a função **vacina()**, que:

C6.1) Chama a função **print_info(cidadao)** com a informação recebida na **resposta** do **processo Servidor**, que irá imprimir a informação completa sobre o cidadão que vai ser vacinado;

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

@@Success@@: C6.1) Dados completos sobre o cidadão a ser vacinado

C6.2) Chama novamente a função **espera_resposta_servidor()**, que espera uma nova resposta do **processo Servidor** (na fila de mensagens com o **tipo = PID_Cidadao**) e preenche a mensagem enviada pelo **processo Servidor** na variável global **resposta**; em caso de erro, afixa uma mensagem de erro e termina o processo com *exit status 1*.

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

@@Error@@: Não é possível ler a resposta do servidor

@@Success@@: Servidor enviou resposta

C6.3) O comportamento do **processo Cidadão** agora irá depender da **resposta** enviada pelo **processo Servidor** no campo **status**:

C6.3.1) Se **status=TERMINADA**, imprime uma mensagem, e termina com *exit status 0*;

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

@@Success@@: C6.3.1) Utente <num_utente>, <nome> vacinado com sucesso

C6.3.2) Se **status=CANCELADA**, imprime uma mensagem de erro, e termina com *exit status 1*;

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

@@Error@@: C6.3.2) O servidor cancelou a vacinação em curso

C7) O sinal **SIGINT** foi armado para que, quando o utilizador interromper o **processo Cidadão** com **<CTRL+C>**, chame a função **cancela_pedido()**. Implemente esta função tal que:

C7.1) Escreva no ecrã uma mensagem;

Outputs esperados (os itens entre <> deverão ser substituídos pelos valores correspondentes):

@@Success@@: C7.1) O cidadão cancelou a vacinação no processo <PID_Cidadao>

C7.2) Altera a variável global **mensagem**, tornando **pedido = CANCELAMENTO**. Chama a função **envia_mensagem_servidor()**, que envia a **mensagem** para a fila de mensagens; em caso de erro no envio, afixa uma mensagem de erro e termina com *exit status 1*;

Outputs esperados (os itens entre <> deverão ser substituídos pelos valores correspondentes):

@@Error@@: Não é possível enviar mensagem para o servidor

@@Success@@: Mensagem para o servidor enviada

C7.3) Chama novamente a função **espera_resposta_servidor()**, que espera e preenche a variável global **resposta** com a nova resposta do **processo Servidor** (na fila de mensagens com o **tipo = PID_Cidadao**);

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

@@Error@@: Não é possível ler a resposta do servidor

@@Success@@: Servidor enviou resposta

C7.4) O comportamento do **processo Cidadão** agora irá depender da **resposta** enviada pelo **processo Servidor**, no campo **status**:

C7.4.1) Se o estado for **CANCELADA**, imprime mensagem, e termina com *exit status 0*;

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

@@Error@@: 7.4.1) Servidor confirmou cancelamento

C7.4.2) Se o estado for **TERMINADA**, imprime mensagem sucesso, termina com *exit status 0*;

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

@@Success@@: 7.4.2) A vacinação já tinha sido concluída

servidor.c

O **processo Servidor** é responsável pela atribuição de um enfermeiro para administrar as vacinas aos **cidadãos** que chegam aos Centros de Saúde. Para efeitos de segurança e evitar o contágio, apenas poderão ser dadas no máximo NUM_VAGAS (valor definido em **common.h**) vacinas em simultâneo.

O **processo Servidor** é responsável pelas seguintes tarefas:

S1) Implemente a função **init_ipc()**, que tenta criar:

- uma fila de mensagens IPC;
- um array de semáforos IPC de dimensão 1;
- uma memória partilhada IPC de dimensão suficiente para conter um elemento Database.

Todos estes elementos têm em comum serem criados com a KEY **IPC_KEY** definida em **common.h** (alterar esta KEY para ter o valor do nº do aluno, como indicado nas aulas), e com **permissões 0600**. Se qualquer um destes elementos IPC já existia anteriormente, dá erro e termina com *exit status 1*. Esta função, em caso de sucesso, preenche as variáveis globais respetivas **msg_id**, **sem_id**, e **shm_id**;

O semáforo em questão será usado com o padrão “Mutex”, pelo que será iniciado com o valor 1;

Outputs esperados (os itens entre <> deverão ser substituídos pelos valores correspondentes):

```
@@Error@@: S1) Fila de Mensagens com a Key definida já existe ou não pode ser criada
@@Error@@: S1) Semáforo com a Key definida já existe ou não pode ser criado
@@Error@@: S1) Semáforo com a Key definida não pode ser iniciado com o valor 1
@@Error@@: S1) Memória Partilhada com a Key definida já existe ou não pode ser criada
@@Success@@: S1) Criados elementos IPC com a Key <IPC_KEY>: MSG <msg_id>, SEM <sem_id>, SHM <shm_id>
```

S2) Chama a função **init_database()**, que inicia a base de dados:

- Associa a variável global **db** com o espaço de Memória Partilhada alocado para **shm_id**; se não o conseguir, dá erro e termina com *exit status 1*;
- Lê o ficheiro **FILE_CIDADAOS** e armazena o seu conteúdo na base de dados usando a função **read_binary()**, assim preenchendo os campos **db->cidadaos** e **db->num_cidadaos**. Se não o conseguir, dá erro e termina com *exit status 1*;
- Lê o ficheiro **FILE_ENFERMEIROS** e armazena o seu conteúdo na base de dados usando a função **read_binary()**, assim preenchendo os campos **db->enfermeiros** e **db->num_enfermeiros**. Se não o conseguir, dá erro e termina com *exit status 1*;
- Inicia a Base de Dados de Vagas, **db->vagas**, colocando o campo **index_cidadao** de todos os elementos com o valor -1.

Outputs esperados (os itens entre <> deverão ser substituídos pelos valores correspondentes):

```
@@Error@@: S2) Erro a ligar a Memória Dinâmica ao projeto
@@Error@@: Erro na abertura do ficheiro
@@Error@@: Erro na leitura do ficheiro
@@Success@@: S2) Base de dados carregada com <num_cidadaos> cidadãos e
               <num_enfermeiros> enfermeiros
```

- S3)** Implemente a função `espera_mensagem_cidadao()`, que espera uma **mensagem** (na fila de mensagens com o **tipo = 1**) e preenche a variável global **mensagem**, assim como preenche o tipo da **resposta** com o **PID_cidadao** recebido. Em caso de erro, termina com erro e *exit status 1*;

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

@@Error@@: Não é possível ler a mensagem do Cidadão

@@Success@@: Cidadão enviou mensagem

- S4)** O comportamento do **processo Servidor** agora irá depender da **mensagem** enviada pelo **processo Cidadão** no campo **pedido**. Implemente a função `trata_mensagem_cidadao()`, que:

- S4.1)** Se o pedido for **PEDIDO**, imprime uma mensagem e avança para o passo **S5**;

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

@@Success@@: S4.1) Novo pedido de vacinação de <PID_cidadao>: <num_utente>, <nome>

- S4.2)** Se o estado for **CANCELAMENTO**, imprime uma mensagem, e avança para o passo **S10**;

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

@@Success@@: S4.2) Cancelamento de vacinação de <PID_cidadao>: <num_utente>, <nome>

- S5)** Implemente a função `processa_pedido()`, que processa um pedido e envia uma **resposta** ao **processo Cidadão**. Para tal, essa função faz vários checks, atualizando o campo **status** da **resposta**:

- S5.1)** Procura o **num_utente** e **nome** na base de dados (BD) de Cidadãos:

- Se o utilizador (Cidadão) não for encontrado na BD Cidadãos → **status = DESCONHECIDO**;
- Se o utilizador (Cidadão) for encontrado na BD Cidadãos, os dados do cidadão deverão ser copiados da BD Cidadãos para o campo **cidadao** da **resposta**;
- Se o Cidadão na BD Cidadãos tiver **estado_vacinacao = 2** → **status = VACINADO**;
- Se o Cidadão na BD Cidadãos tiver **PID_cidadao > 0** → **status = EMCURSO**; caso contrário, afeta o **PID_cidadao** da BD Cidadãos com o valor do **PID_cidadao** da **mensagem**;

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

@@Error@@: S5.1) Cidadão <num_utente>, <nome> não foi encontrado na BD Cidadãos

@@Success@@: S5.1) Cidadão <num_utente>, <nome> encontrado,
estado_vacinacao=<estado_vacinacao>, status=<status>

- S5.2)** Caso o Cidadão esteja em condições de ser vacinado (i.e., se **status não for DESCONHECIDO, VACINADO nem EMCURSO**), procura o enfermeiro correspondente na BD Enfermeiros:

- Se não houver centro de saúde, ou não houver nenhum enfermeiro no centro de saúde correspondente → **status = NAOHAENFERMEIRO**;
- Se há enfermeiro, mas este não tiver disponibilidade → **status = AGUARDAR**.

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

@@Error@@: S5.2) Enfermeiro do CS <localidade> não encontrado na BD Enfermeiros

@@Success@@: S5.2) Enfermeiro do CS <localidade> encontrado,
disponibilidade=<disponibilidade>, status=<status>

S5.3) Caso o enfermeiro esteja disponível, procura uma vaga para vacinação na BD Vagas. Para tal, chama a função `reserva_vaga(index_cidadao, index_enfermeiro)` usando os índices do Cidadão e do Enfermeiro nas respetivas BDs:

- Se essa função tiver encontrado e reservado um index **vaga_ativa** → **status = OK**;
- Se essa função não conseguiu encontrar uma vaga livre → **status = AGUARDAR**.

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

@@Error@@: S5.3) Não foi encontrada nenhuma vaga livre para vacinação

@@Success@@: S5.3) Foi reservada a vaga <vaga_ativa> para vacinação, status=<status>

S5.4) Se no final de todos os checks, **status = OK**, chama a função `vacina()`, caso contrário, chama a função `envia_resposta_cidadao()`, que **envia a resposta de erro** ao Cidadão;

S6) Implemente a função `vacina()`, que processa a vacinação.

S6.1) Cria um processo filho através da função `fork()`;

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

@@Error@@: S6.1) Não foi possível criar um novo processo

@@Success@@: S6.1) Criado um processo filho com PID_filho=<PID_filho>

S6.2) O processo filho chama a função `servidor_dedicado()`;

S6.3) O processo pai regista o process ID do processo filho no campo `PID_filho` na BD de Vagas com o índice da variável global **vaga_ativa**;

S7) Implemente a função `servidor_dedicado()`, que define o comportamento do servidor dedicado:

S7.1) Arma o sinal `SIGTERM`;

S7.2) Envia a resposta para o Cidadão, chamando a função `envia_resposta_cidadao()`. Implemente também esta função, que envia a mensagem **resposta** para o cidadão, contendo os dados do Cidadão preenchidos em **S5.1** e o campo **status = OK**;

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

@@Error@@: Não é possível enviar resposta para o cidadão

@@Success@@: Resposta para o cidadão enviada

S7.3) Coloca a disponibilidade do enfermeiro afeto à **vaga_ativa** com o valor 0 (Indisponível);

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

@@Success@@: S7.3) Enfermeiro associado à vaga <vaga_ativa> indisponível

S7.4) Imprime uma mensagem e aguarda (em espera passiva!) **TEMPO_CONSULTA** segundos;

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

@@Success@@: S7.4) Vacina em curso para o cidadão <num_utente>, <nome>, e com o enfermeiro <ced_profissional>, <nome> na vaga <vaga_ativa>

S7.5) Envia nova resposta para o Cidadão, chamando a função `envia_resposta_cidadao()` contendo os dados do Cidadão preenchidos em **S5.1** e o campo **status=TERMINADA**, para indicar que a consulta terminou com sucesso;

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

`@@Error@@`: Não é possível enviar resposta para o cidadão

`@@Success@@`: Resposta para o cidadão enviada

S7.6) Atualiza os dados Cidadão (**PID_cidadao=-1** e incrementa **estado_vacinacao**) na BD Cidadãos, e enfermeiro (**disponibilidade=1** e incrementa **nr_vacinas_dadas**) na BD de Enfermeiros;

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

`@@Success@@`: S7.6) Cidadão atualizado na BD para `estado_vacinacao=<estado_vacinacao>`, Enfermeiro atualizado na BD para `nr_vacinas_dadas=<nr_vacinas_dadas>` e `disponibilidade=<disponibilidade>`

S7.7) Liberta a vaga **vaga_ativa** da BD de Vagas, invocando a função `liberta_vaga(vaga_ativa)`;

S7.8) Termina o processo Servidor Dedicado (filho) com `exit status 0`.

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

`@@Success@@`: S7.8) Servidor dedicado Terminado

S8) Implemente a função `reserva_vaga(index_cidadao, index_enfermeiro)`, que tentará reservar uma vaga livre na BD de Vagas. Para tal:

S8.1) Procura uma vaga livre (`index_cidadao < 0`) na BD de Vagas. Se encontrar uma entrada livre:

S8.1.1) Atualiza o valor da variável global **vaga_ativa** com o índice da vaga encontrada;

S8.1.2) Atualiza a entrada de Vagas **vaga_ativa** com o índice do cidadão e do enfermeiro

S8.1.3) Retorna o valor do índice de vagas **vaga_ativa** ou **-1** se não encontrou nenhuma vaga

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

`@@Error@@`: S8.1.3) Não foi encontrada nenhuma vaga livre

`@@Success@@`: S8.1.3) Foi reservada a vaga livre com o index <vaga_ativa>

S9) Implemente a função `liberta_vaga(index_vaga)`, que liberta a vaga da BD de Vagas, colocando o campo **index_cidadao** dessa entrada da BD de Vagas com o valor **-1**.

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

`@@Success@@`: S9) A vaga com o index <index_vaga> foi libertada

S10) Implemente a função `cancela_pedido()`, que processa o cancelamento de um pedido de vacinação e envia uma resposta ao `processo Cidadão`. Para este efeito, a função:

S10.1) Procura na BD de Vagas a vaga correspondente ao Cidadão em questão (procura por `index_cidadao`). Se encontrar a entrada correspondente, obtém o PID_filho do Servidor Dedicado correspondente;

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

@@Error@@: S10.1) Não foi encontrada nenhuma sessão de vacinação com o cidadão <num_utente>, <nome>

@@Success@@: S10.1) Foi encontrada a sessão do cidadão <num_utente>, <nome> na sala com o index <index_vaga>

S10.2) Envia um sinal **SIGTERM** ao `processo Servidor Dedicado` (filho) que está a tratar da vacinação;

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

@@Success@@: S10.2) Enviado sinal SIGTERM ao Servidor Dedicado com PID=<PID_filho>

S11) Implemente a função `termina_servidor()`, que irá tratar do fecho do servidor, e que:

S11.1) Envia um sinal **SIGTERM** a todos os `processos Servidor Dedicado` (filhos) ativos;

S11.2) Grava o ficheiro `FILE_ENFERMEIROS`, usando a função `save_binary()`;

S11.3) Grava o ficheiro `FILE_CIDADAOS`, usando a função `save_binary()`;

S11.4) Remove do sistema (IPC Remove) os semáforos, a Memória Partilhada e a Fila de Mensagens.

S11.5) Termina o processo servidor com `exit status 0`.

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

@@Success@@: S11.4) Servidor Terminado

S12) Implemente a função `termina_servidor_dedicado()`, que irá tratar do fecho do servidor dedicado, e que:

S12.1) Envia a resposta para o Cidadão, chamando a função `envia_resposta_cidadao()` com o campo `status=CANCELADA`, para indicar que a consulta foi cancelada;

S12.2) Liberta a `vaga_ativa` da BD de Vagas, invocando a função `liberta_vaga(vaga_ativa)`;

S12.3) Termina o processo do servidor dedicado com `exit status 0`;

Outputs esperados (itens entre <> substituídos pelos valores correspondentes):

@@Success@@: S12.3) Servidor Dedicado Terminado

NOTA Importante:

Apesar de não estar explicitamente indicado no enunciado, os alunos já conhecem os problemas relativos a exclusão mútua no acesso à Memória Partilhada, pelo que deverão tomar as devidas precauções para que não haja conflitos no acesso à mesma entre o Servidor e os Servidores Dedicados. Relembra-se que a garantia de exclusão deverá ocupar o menor número de tempo possível, para permitir a maior concorrência possível entre os processos a aceder às zonas críticas.

Para tal, os alunos podem utilizar o semáforo **Mutex** que está disponibilizado no código, que é operado utilizando as seguintes funções:

- `sem_mutex_down()`: Faz lock ao semáforo Mutex
- `sem_mutex_up()`: Faz unlock ao semáforo Mutex

O desafio é saber em que altura, onde, e como usar as funções acima definidas, sabendo que o *locking* deverá ser feito durante o menor tempo possível, mas que não pode comprometer a zona crítica para garantir que o servidor principal e os dedicados não entram em conflito.

Boa sorte!!!