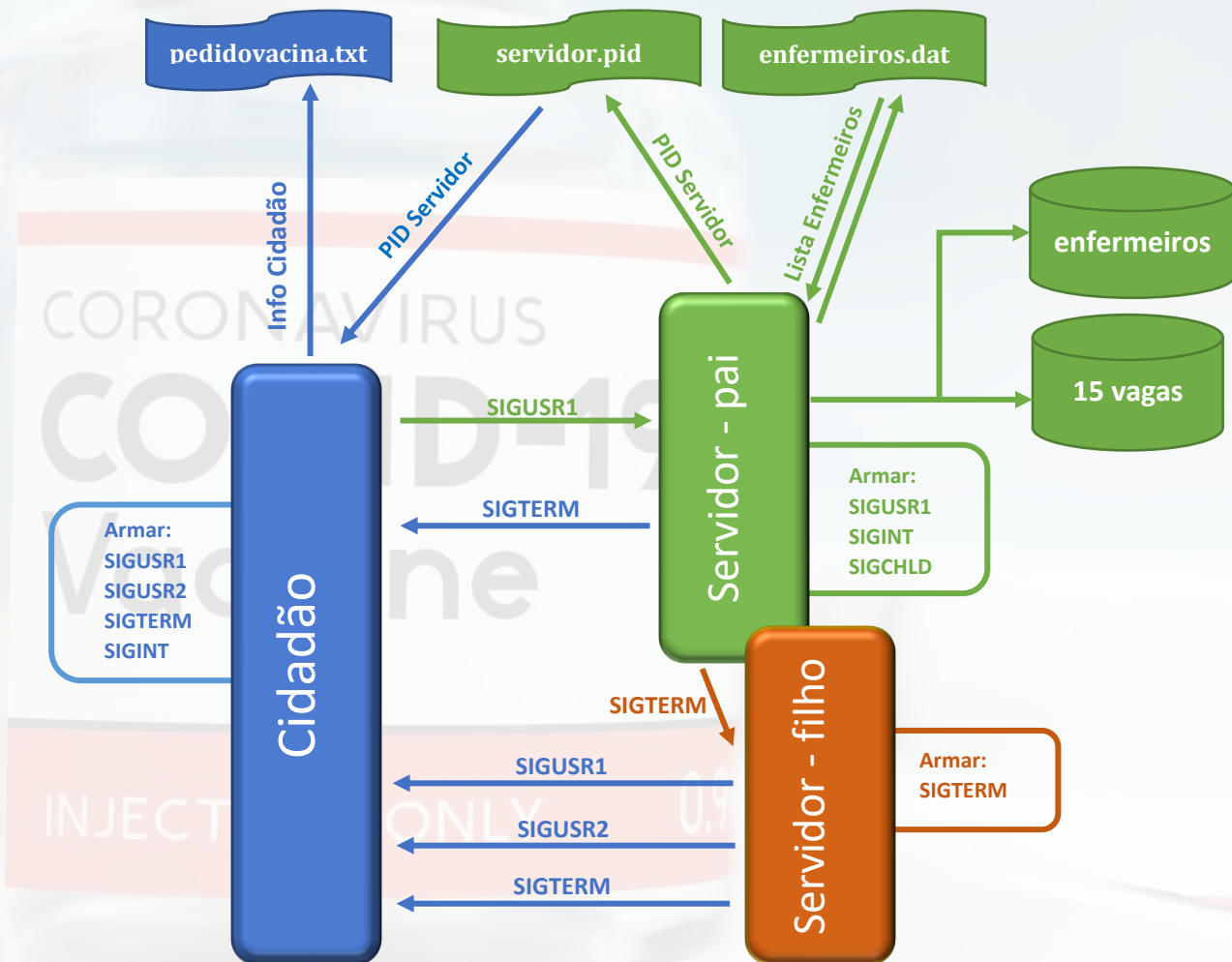


Projeto Covid-IUL (Parte 2)

Nesta parte do trabalho, será implementado um modelo simplificado da Administração de vacinas dos cidadãos do sistema Covid-IUL, baseado em comunicação por sinais 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**.

Entrega, relatório e autoavaliação

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

A entrega da Parte 2 do trabalho será realizada através da criação de **um** ficheiro ZIP cujo nome é o nº do aluno, e.g., **“a<nºaluno>-parte-2.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 primeira parte, apenas os ficheiros Shell (*.sh). Cada um dos módulos será desenvolvido com base nos ficheiros fornecidos, e que estão na diretoria do Tigre **“/home/so/trabalho-2020-2021/parte-2”**, 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-2.zip <ficheiros>**, por exemplo:

```
$ zip a123456-parte-2.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 2”;
- Dentro do formulário “Visualizar Exercício de carregamento: Trabalho Prático 2020/2021 Parte 2”, 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 II – Processos e Sinais

Data de entrega: **25 de abril de 2021**

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

- **“pedidovacina.txt”**: Ficheiro de texto com os dados do cidadão a vacinar, com todos os valores numa única linha separados por dois pontos (“:”), na forma:
“num_utente:nome:idade:localidade:nºtelemóvel:estado_vacinacao:PID_cidadao”
- **“servidor.pid”**: Ficheiro de texto com o PID do servidor de enfermeiros. O ficheiro deve conter apenas o PID, e.g., “1233445”;
- **“enfermeiros.dat”**: Ficheiro binário com a informação sobre os enfermeiros fornecido pelo sistema (copiar para a vossa diretoria a partir da diretoria existente no servidor Tigre /home/so/trabalho-2020-2021/parte-2/enfermeiros.dat). **Os alunos deverão assumir que este ficheiro pode vir a ter outros conteúdos ou outro tamanho.** A estrutura do ficheiro é **Enfermeiro**.

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

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

(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

enfermeiros.dat: Ficheiro de elementos do tipo **Enfermeiro** (dados em baixo).

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

(em formato legível)

enfermeiros.dat

12345	Ana Correia	CSLisboa	0	1
98765	Joao Vieira	CSPorto	0	1
54321	Joana Pereira	CSSantarem	0	1
65432	Jorge Vaz	CSSetubal	0	1
76543	Diana Almeida	CSLeiria	0	1
87654	Diogo Cruz	CSBraga	0	1
32198	Bernardo Pato	CSBraganca	0	1
21654	Maria Mendes	CSBeja	0	1
88888	Alice Silva	CSEvora	0	1
96966	Gustavo Carvalho	CSFaro	0	1

```
typedef struct {  
    int index_enfermeiro;  
    Cidadao cidadao;  
    int PID_filho;  
} Vaga;
```

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. Todos os passos deverão mostrar as mensagens de sucesso ou erro indicadas, usando as macros acima mencionadas.

Assim, definem-se as seguintes tarefas a desenvolver:

- C1)** Pede ao Cidadão (utilizador) os seus dados para que seja feita a admissão, nomeadamente o número de utente, nome, idade, localidade e nº telemóvel, obrigatoriamente nessa ordem. O estado da vacinação começa sempre com valor 0;

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

@@Success@@: C1) Dados Cidadão: <num_utente>; <nome>; <idade>; <localidade>; <nºtelemóvel>; 0

- C2)** Cria um elemento do tipo **Cidadao** com as informações sobre o cidadão, preenchendo o campo <PID_cidadao> com o PID deste **processo Cidadão**;

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

@@Success@@: C2) PID Cidadão: <PID Cidadão>

- C3)** Verifica se o ficheiro de texto **pedidovacina.txt** já existe. Caso o ficheiro **pedidovacina.txt** já exista, isso significa que um outro cidadão já iniciou a submissão do processo de vacinação. Nesse caso, mostra uma mensagem de erro no ecrã, e termina o **processo Cidadão**, ou mensagem de sucesso;

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

@@Error@@: C3) Não é possível iniciar o processo de vacinação neste momento

@@Success@@: C3) Ficheiro FILE_PEDIDO_VACINA pode ser criado

- C4)** Não existindo o ficheiro **pedidovacina.txt**, então cria esse ficheiro onde regista as informações desse elemento **Cidadao** no ficheiro **pedidovacina.txt**, com a forma:

“num_utente:nome:idade:localidade:nºtelemóvel:estado_vacinacao:PID_cidadao”

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

@@Error@@: C4) Não é possível criar o ficheiro FILE_PEDIDO_VACINA

@@Success@@: C4) Ficheiro FILE_PEDIDO_VACINA criado e preenchido

- C5)** Arma e trata o sinal **SIGINT** (só pode acontecer depois do passo **C4**) para que, no caso de o utilizador interromper o **processo Cidadão** com o atalho <CTRL+C>, escreve no ecrã a mensagem “O cidadão cancelou a vacinação, o pedido nº <PID_cidadao> foi cancelado”, apaga o ficheiro **pedidovacina.txt** e termina o **processo Cidadão**;

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

@@Success@@: C5) O cidadão cancelou a vacinação, o pedido nº <PID> foi cancelado

C6) Lê o PID do **processo Servidor** do ficheiro **servidor.pid** (se o mesmo existir) e envia um sinal **SIGUSR1** ao **processo Servidor** para que este lhe indique se pode ou não ser vacinado;

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

@@Error@@: C6) Não existe ficheiro FILE_PID_SERVIDOR!

@@Success@@: C6) Sinal enviado ao Servidor: <PID Servidor>

C7) Arma e trata o sinal **SIGUSR1**, para o caso do **processo Servidor** indicar que existe enfermeiro disponível e, portanto, a vacina vai ser administrada. Se receber esse sinal, escreve no ecrã a mensagem “Vacinação do cidadão com o pedido nº <PID_cidadao> em curso.”, e apaga o ficheiro **pedidovacina.txt**;

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

@@Success@@: C7) Vacinação do cidadão com o pedido nº <PID Cidadão> em curso

C8) Arma e trata o sinal **SIGUSR2**, para o caso do **processo Servidor** indicar que a vacinação terminou. Se receber esse sinal, escreve no ecrã a mensagem “Vacinação do cidadão com o pedido nº <PID_cidadao> concluída.”, e termina o **processo Cidadão**;

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

@@Success@@: C8) Vacinação do cidadão com o pedido nº <PID Cidadão> concluída

C9) Arma e trata o sinal **SIGTERM**, para a eventualidade de receber a indicação do **processo Servidor** de que não é possível realizar a vacinação. Se receber esse sinal, escreve no ecrã a mensagem “Não é possível vacinar o cidadão no pedido nº <PID_cidadao>”. Apaga o ficheiro **pedidovacina.txt** e termina o **processo Cidadão**;

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

@@Success@@: C9) Não é possível vacinar o cidadão no pedido nº <PID Cidadão>

C10) (Extra-points) No ponto **C3**, no caso do ficheiro **pedidovacina.txt** já existir, escreve a mensagem de erro indicada, mas em vez do **processo Cidadão** terminar, arma o sinal **SIGALRM**, e espera 5 segundos em pausa, após o que volta a tentar novamente, e assim sucessivamente, até que o ficheiro **pedidovacina.txt** deixe de existir, altura em que prossegue para o passo **C4**.

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

@@Error@@: C3) Não é possível iniciar o processo de vacinação neste momento

@@Success@@: C3) Ficheiro FILE_PEDIDO_VACINA pode ser criado

servidor.c

O **processo Servidor** é responsável pela atribuição de um enfermeiro para administrar as vacinas aos **cidadãos** que chegam aos Centros de Saúde. Este módulo estará sempre ativo, à espera da chegada de **cidadãos**.

Para efeitos de segurança e evitar o contágio, apenas poderão ser dadas no máximo NUM_VAGAS (valor definido em **common.h**) vacinas em simultâneo. Este módulo mantém uma lista com NUM_VAGAS **vagas**:

```
Vaga vagas[NUM_VAGAS];
```

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

S1) Regista (se conseguir) o PID do seu **processo Servidor** no ficheiro **servidor.pid**;

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

```
@@Error@@: S1) Não consegui registar o servidor!
```

```
@@Success@@: S1) Escrevi no ficheiro FILE_PID_SERVIDOR o PID: <PID Servidor>
```

S2) Define uma estrutura de dados dinâmica em memória, **enfermeiros** com o tamanho certo para comportar toda a lista de enfermeiros, sendo preenchida com os dados do ficheiro **enfermeiros.dat**. Este ficheiro poderá ter outro tamanho e conteúdos diferentes do fornecido;

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

```
@@Error@@: S2) Não consegui ler o ficheiro FILE_ENFERMEIROS!
```

```
@@Success@@: S2) Ficheiro FILE_ENFERMEIROS tem <size> bytes, ou seja, <nº enfermeiros> enfermeiros
```

S3) Inicia a lista de vagas com o campo-chave **index_enfermeiro** com o valor “-1”. Isto equivale a “limpar” a lista;

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

```
@@Success@@: S3) Iniciei a lista de <Nº Vagas> vagas
```

S4) Arma e trata o sinal **SIGUSR1** para que sejam tratados os **cidadãos** que chegam aos Centros de Saúde;

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

```
@@Success@@: S4) Servidor espera pedidos
```

S5) Fica à espera de receber pedidos de vacinação (**SIGUSR1**). Quando receber este sinal:

S5.1) Lê a informação do **cidadão** no ficheiro **pedidovacina.txt** e verifica qual o Centro de Saúde desse **cidadão**, escrevendo no ecrã a mensagem “Chegou o cidadão com o pedido nº <PID_cidadao>, com nº utente <num_utente>, para ser vacinado no Centro de Saúde <CSlocalidade>”. Não faz parte do âmbito a validação dos dados do ficheiro, está na forma:

“num_utente:nome:idade:localidade:nºtelemóvel:estado_vacinacao:PID_cidadao”

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

@@Error@@: S5.1) Não foi possível abrir o ficheiro FILE_PEDIDO_VACINA

@@Error@@: S5.1) Não foi possível ler o ficheiro FILE_PEDIDO_VACINA

@@Success@@: S5.1) Dados Cidadão: <num_utente>; <nome>; <idade>; <localidade>; <nºtelemóvel>; 0

S5.2) Verifica na estrutura **enfermeiros** se o enfermeiro correspondente ao Centro de Saúde do **cidadão** está disponível (campo **disponibilidade** com o valor **1**):

S5.2.1) Se o enfermeiro **não estiver disponível**, dá uma mensagem de erro, e envia um sinal **SIGTERM** ao **processo PID_cidadao** que informa o **cidadão** que o processo de vacinação não é possível de momento, e **retorna ao ponto S5** à espera de novos pedidos;

Outputs esperados (os itens entre <> são substituídos pelos valores respetivos):

@@Error@@: S5.2.1) Enfermeiro <Index Enfermeiro> indisponível para o pedido <PID Cidadão> para o Centro de Saúde <CS_enfermeiro>

@@Success@@: S5.2.1) Enfermeiro <Index Enfermeiro> disponível para o pedido <PID Cidadão>

S5.2.2) Se o enfermeiro do CS **estiver disponível**, valida se há vagas na lista de Vagas para vacinação neste momento. Se **não houver vaga**, dá uma mensagem de erro, e envia um sinal **SIGTERM** ao **processo PID_cidadao** que informa o **cidadão** que o processo de vacinação não é possível de momento, e **retorna ao ponto S5** à espera de novos pedidos.

Outputs esperados (os itens entre <> são substituídos pelos valores respetivos):

@@Error@@: S5.2.2) Não há vaga para vacinação para o pedido <PID Cidadão>

@@Success@@: S5.2.2) Há vaga para vacinação para o pedido <PID Cidadão>

S5.3) Se o enfermeiro **estiver disponível** e **houver “vaga”**, preenche a entrada na lista **vagas** com o índice correspondente da lista de enfermeiros, a informação submetida do **Cidadao**, e altera a **disponibilidade** desse enfermeiro na lista de enfermeiros para **0 (indisponível)**;

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

@@Success@@: S5.3) Vaga nº <Nº Vaga> preenchida para o pedido <PID Cidadão>

S5.4) Em seguida, o **processo Servidor** cria um **processo Servidor-Filho** (fork) que será responsável pela vacinação daquele cidadão;

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

@@Error@@: S5.4) Não foi possível criar o servidor dedicado

@@Success@@: S5.4) Servidor dedicado <PID Filho> criado para o pedido <PID Cidadão>

S5.5) Após a criação do **processo Servidor-Filho**, o **processo Servidor-Pai**:

S5.5.1) Atualiza a entrada na lista de Vagas preenchida anteriormente com o **Cidadao** para incluir também a informação do **PID desse processo Servidor-Filho**;

Outputs esperados (os itens entre <> são substituídos pelos valores respetivos):

@@Success@@: S5.5.1) Servidor dedicado <PID Filho> na vaga <Nº Vaga>

S5.5.2) O **processo Servidor-Pai** NÃO deverá ficar à espera que o processo da vacinação (**processo Servidor-Filho**) termine, e deve **retornar ao ponto S5** à espera de novos pedidos. Deve sim armar o sinal **SIGCHLD** para “acordar” quando o **processo Servidor-Filho** terminar;

Outputs esperados (os itens entre <> são substituídos pelos valores respetivos):

@@Success@@: S5.5.2) Servidor aguarda fim do servidor dedicado <PID Filho>

S5.5.3) Quando, finalmente, o **processo Servidor-Filho** terminar, o **processo Servidor-Pai** será notificado, e nessa altura:

S5.5.3.1) Deve “limpar” a entrada correspondente da tabela de vagas;

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

@@Success@@: S5.5.3.1) Vaga <Nº Vaga> que era do servidor dedicado <PID Filho> libertada

S5.5.3.2) Atualiza o perfil do enfermeiro que deu a vacina como disponível (**disponibilidade = 1**);

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

@@Success@@: S5.5.3.2) Enfermeiro <Index Enfermeiro> atualizado para disponível

S5.5.3.3) Incrementa o nº de vacinas dadas por esse enfermeiro (**num_vac_dadas**);

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

@@Success@@: S5.5.3.3) Enfermeiro <Index Enfermeiro> atualizado para <nº Vacinas> vacinas dadas

S5.5.3.4) Atualiza o ficheiro **enfermeiros.dat**, apenas com a informação desse enfermeiro (e só desse) – ou seja, atualiza no ficheiro **enfermeiros.dat** o nº de vacinas dadas pelo enfermeiro;

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

@@Success@@: S5.5.3.4) Ficheiro FILE_ENFERMEIROS <Index Enfermeiro> atualizado para <nº Vacinas> vacinas dadas

S5.5.3.5) Volta ao que estava a fazer antes do **processo Servidor-Filho** terminar.

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

@@Success@@: S5.5.3.5) Retorna

S5.6) Por outro lado, após a criação do **processo Servidor-Filho**, o **processo Servidor-Filho**:

S5.6.1) Arma o sinal **SIGTERM** que poderá ser enviado pelo **processo Servidor-Pai** em caso de terminação do mesmo. Se receber este sinal, deverá mostrar uma mensagem informativa, envia um sinal **SIGTERM** ao **processo PID_cidadao** que informa o **cidadão** que o processo de vacinação não é possível de momento, e termina o **processo Servidor-Filho**;

Outputs esperados (os itens entre <> são substituídos pelos valores respetivos):

@@Success@@: S5.6.1) SIGTERM recebido, servidor dedicado termina Cidadão

S5.6.2) Envia um sinal do tipo **SIGUSR1** ao **processo PID_cidadao**, informando o cidadão de que a sua vacinação começou;

Outputs esperados (os itens entre <> são substituídos pelos valores respetivos):

@@Success@@: S5.6.2) Servidor dedicado inicia consulta de vacinação

S5.6.3) Espera **TEMPO_CONSULTA** segundos (valor definido no ficheiro **common.h**) correspondente à duração da vacinação, e escreve no ecrã "Vacinação terminada para o cidadão com o pedido nº <PID_cidadao>;

Outputs esperados (os itens entre <> são substituídos pelos valores respetivos):

@@Success@@: S5.6.3) Vacinação terminada para o cidadão com o pedido nº <PID Cidadão>

S5.6.4) No final do processo de vacinação, envia um sinal do tipo **SIGUSR2** ao **processo PID_cidadao** e termina o **processo Servidor-Filho**.

Outputs esperados (os itens entre <> são substituídos pelos valores respetivos):

@@Success@@: S5.6.4) Servidor dedicado termina consulta de vacinação

S6) O **processo Servidor** deve armar e tratar o sinal **SIGINT**, para que possa ser encerrado pelo utilizador com o atalho <CTRL+C>. Quando isso acontecer, deverá enviar um sinal **SIGTERM** a todos os **processos Servidor-Filho** existentes para os terminar, remove o ficheiro **servidor.pid** e termina o **processo Servidor**. Mesmo que esse atalho <CTRL+C> aconteça durante uma ou mais consultas, mais nenhuma vacinação ocorre.

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

@@Success@@: S6) Servidor terminado

Boa sorte!!!