INSTITUTO UNIVERSITÁRIO DE LISBOA

Departamento de Ciências e Tecnologias da Informação

Projeto Cliniq-IUL

Henrique Eduardo Rodrigues da Cunha Fonseca
(ET-B4 – 94089)

Relatório do Projeto Cliniq-IUL (Parte 2) da UC Sistemas Operativos

Ano Letivo 2020/2021

Novembro, 2020

Parte II – Processos e Sinais

O presente relatório via apresentar informações sobre a elaboração da **Parte 2** do projeto **Clinic-IU**L da **UC Sistemas Operativos**. Nesta fase do trabalho foi criado um conjunto de **2 ficheiros** escritos na **linguagem de programação C**, para implementar um modelo simplificado de triagem e consultas de pacientes no sistema **Cliniq-IUL**, baseado em **comunicaçã**o por **sinais** entre **processos**. Para simular a realização de consultas no sistema Cliniq-IUL existem **2 módulos** – **Cliente** e **Servidor**.

Cliente.c

O módulo **Cliente.c** é responsável pelo encaminhamento dos pacientes. Este módulo será utilizado para solicitar o encaminhamento dos pacientes para as suas consultas. Após esta indicação, o paciente é "encaminhado" e, caso haja disponibilidade, é realizada a consulta, ficando este módulo a aguardar até que a mesma termine.

Neste módulo existem **3 funções**, duas delas para **obter** as **informações** sobre a **consulta** e **criar** um **objeto** do **tipo** "**Consulta**" com as informações obtidas, ***getAppointmentInformation()*** e ***createAppointment(),*** **respetivamente**; e uma outra função para **enviar** um sinal **SIGUSR1** ao módulo **Servidor.c**, para que este dê **início** à **consulta**, ***sendBeginAppointmentRequest()***. Uma vez que não faria muito sentido o módulo **Cliente.c terminar** o próprio processo enquanto a **consulta** está **a decorrer**, existe a função **ignoreSIGINT()** que **ignora** o sinal **SIGINT** enquanto a consulta está a decorrer (o sinal SIGINT é apenas ignorado enquanto a consulta está a decorrer, fora isso termina o processo como suposto). Existem ainda mais 4 funções - ***appointmentCanceled()***, ***appointmentInitiated()***, ***appointmentConcluded()*** e ***appointmentInvalid()*** – para **tratar** os **sinais SIGINT**, **SIGHUP**, **SIGTERM** e **SIGUSR2**, respetivamente. Na função ***getAppointmentInformation()*** existe um ciclo ***while*** para **validação** do **input** tipo de consulta (tem de ser um inteiro entre 1 a 3), que imprime uma mensagem de erro se o input for inválido. Se for inserido um input que não seja do tipo *int* o *loop* não funciona e, de forma errada, imprime infinitamente a mensagem de erro (não consegui resolver este erro, penso que seja pelo facto de usar a função *scanf()* para a validação, uma solução poderia ser usar em alternativa a função *fgets()*). Se o módulo **Servidor.c terminar** enquanto a **consulta** estiver **a decorrer**, esta **recebe** um sinal **SIGQUIT**, e o *handler* ***serverShutDown()*** imprime uma **mensagem de erro** e **termina o processo**.

A **alínea extra** diz o seguinte: "Como pode ter já percebido, se houver um cliente que faz um novo pedido antes do anterior ter sido satisfeito, o ficheiro PedidoConsulta.txt é escrito por cima com o novo pedido.". Para evitar esta situação, e **permitir** que **várias consultas** possam **decorrer ao mesmo tempo** (no máximo 10), foi implementada uma **solução em substituição à alínea extra** (C8). Esta **implementação** foi feita no módulo **Servidor.c** e explicada na secção referente ao mesmo.

Servidor.c

O módulo **Servidor de Consultas** é responsável pela realização das consultas que chegam ao sistema Cliniq-IUL. Este módulo está normalmente ativo, à espera de pedidos de consulta. As consultas têm uma duração de 10 segundos. Findo o tempo da consulta, este módulo sinaliza o paciente de que a sua consulta terminou. Este módulo possui **contadores** de **consultas** por **tipo**, e uma **lista** – ***(Consulta) list_appointments[10]*** - com capacidade para agendar 10 consultas.

Neste módulo existem as funções ***initialize()*** e ***registPID()*** para iniciar a **lista de consultas** com o **campo tipo** a **'-1'** e **registar** o **PID** do seu processo no ficheiro **SrvConsultas.pid**, respetivamente. Existem ainda as funções ***initiateAppointmentProcess()*** e ***serverShutDown()*** para **tratar** os sinais **SIGUSR1** e **SIGINT**, respetivamente. Se existirem **consultas a decorrer** quando o **processo recebe** o sinal **SIGINT**, todos os processos filhos (que são criados quando uma consulta é iniciada) terminam, e a função ***serverShutDown()*** **envia** um sinal **SIGQUIT** a cada consulta (processos do tipo Cliente.c) que esteja a decorrer para estas também **terminarem**.

Para permitir que várias **consultas** possam decorrer em **simultâneo** (com um máximo de 10 consultas), foi **implementada** uma **estrutura de dados auxiliar Queue** (*FIFO – first in first out*) – ***(int) roms[10]*** – para **guardar** a **ordem** das **salas** em que as consultas foram **iniciadas**. Tal **necessidade** surge do facto de que consoante a **ordem de chegada** de pedidos de consultas e a ordem de termino das mesmas, pode acontecer uma **consulta** na **sala 0 acabar primeiro** que uma **consulta** na **sala 2**.
Exemplo:
Se estiverem a decorrer 2 consultas do tipo 2, o *array rooms* guardaria 0 e 1 (salas em que as consultas decorrem), se entretanto a consulta na sala 0 acabar e, antes da consulta da sala 1 acabar, chegarem mais 2 pedidos de consultas (do tipo 2), o *array rooms* passaria a guardar os valores 1, 0 e 2 porque a sala disponível para a consulta será, em preferência ascendente, o primeiro índice livre do *array list_appointments*:

<início>
list_appointments = [-1,-1,-1,-1,-1,-1,-1,-1,-1,-1] | rooms = [-1,-1,-1,-1,-1,-1,-1,-1,-1,-1]
<iniciam-se consultas nas salas 0 e 1>
list_appointments = [2,2,-1,-1,-1,-1,-1,-1,-1,-1] | rooms = [0,1,-1,-1,-1,-1,-1,-1,-1,-1]
<termina consulta sala 0>
list_appointments = [-1,2,-1,-1,-1,-1,-1,-1,-1,-1] | rooms = [1,-1,-1,-1,-1,-1,-1,-1,-1,-1]
<iniciaam-se consultas nas salas 1 e 2>
list_appointments = [2,2,2,-1,-1,-1,-1,-1,-1,-1] | rooms = [1,0,2,-1,-1,-1,-1,-1,-1,-1]
<termina a consulta na sala 1>
list_appointments = [2,-1,2,-1,-1,-1,-1,-1,-1,-1] | rooms = [0,2,-1,-1,-1,-1,-1,-1,-1,-1]

<termina a consulta na sala 0>
list_appointments = [-1,-1,2,-1,-1,-1,-1,-1,-1,-1] | rooms = [2,-1,-1,-1,-1,-1,-1,-1,-1,-1]
<termina a consulta na sala 2>
list_appointments = [-1,-1,-1,-1,-1,-1,-1,-1,-1,-1] | rooms = [-1,-1,-1,-1,-1,-1,-1,-1,-1,-1]

Quando uma **consulta** é **iniciada**, o módulo **Servidor.c cria** um **processo filho** onde **decorre** a consulta. Quando esta **acaba**, o processo **filho envia** um sinal **SIGUSR2** ao processo **pai**, significando que a **consulta terminou**. O processo **pai trata** o sinal com a função ***updatelista()*** e esta **remove** a **consulta** do *array* ***list_appointments*** **removendo** também a **sala** da **consulta terminada** no *array* ***rooms*** (fazendo um *left shift* dos valores do *array rooms*).

Consulta.h

Este é um ficheiro auxiliar onde está definida a **estrutura de dados** das **consultas**. Cada consulta tem **3 atributos** – (*int*) tipo; (*char*[100]) descricao e (*int*) pid_consulta.