TECNOLOGIAS
E ARQUITETURA

# Relatório SO – Parte 3 IPCS


# Gabriel Monteiro | Nº 92458 | ETB-2


Na última fase do projeto foi realizado um sistema de Consultas através do desenvolvimento de dois módulos: Cliente e Servidor, com o mesmo objetivo do da 2ª fase, para a continuação do projeto ***Cliniq-IUL*** que tinha por base IPCS's.

## *Cliente.c*

Comecei o desenvolvimento do módulo Cliente por criar a função ***creat_consulta()*** que tem como objetivo criar a consulta através do respetivo tipo e descrição inserido pelo o usuário que está a executar o módulo, à semelhança da última parte. Em seguida, foi criada uma função com o objetivo de enviar ao servidor uma mensagem através da ***mailbox 1*** de uma ***message queue.*** A função chama-se ***sendPedidoConsulta()*** e começa por criar uma variável do tipo Consulta, associando os vários campos desta variável às informações definidas na função que cria a consulta. Após isso temos como objetivo informar o servidor que chegou um novo pedido de consulta. Para isso ligamo-nos à ***message queue*** com a ***IPC_KEY*** definida em ***defines.h***. Em seguida há que definir em ***defines.h*** uma estrutura mensagem, que tem como características um ***long tipo*** e um ***char texto [120]***. Definimos então o tipo da mensagem como 1, uma vez que é pedido que se envie para a mailbox 1, e envio toda a informação da consulta, separa por ' ; ' através de um ***sprintf()*** e definindo-a como o texto da mensagem que será enviada.

Prosseguimos agora para a função ***recieveMessageWithPid()*** que permite ao módulo ***Cliente*** ficar à escuta de mensagens da ***message queue*** com ***mailbox*** correspondente ao ***número do PID.*** Esta função consiste em ligarmo-nos à message queue e receber um vetor de caracteres do módulo ***Servidor*** (no caso do projeto será apenas o status da Consulta), que por sua vez será convertido para um inteiro e acumulado numa variável através do procedimento ***atoi()***. Após recebida uma mensagem com o status da Consulta, do módulo ***Servidor***, iremos verificar qual será e dependendo do seu valor, irá desempenhar funções diferentes. Caso receba um valor inteiro 2, fará o procedimento "***starting_consulta()***" que irá fazer o print da consulta iniciada. Caso seja um valor inteiro 3, fará o procedimento "***terminate_consulta()***", em

que realiza a verificação de se recebeu a mensagem com o valor inteiro 2, e se sim, faz o print da consulta concluída e termina o processo. Por fim, caso o valor inteiro seja 4, irá realizar o procedimento "***decline_consulta()"*** que consiste em dar um print a referir ao utilizador que a consulta não é possível e termina o processo em questão.

Por fim, foi pedido que o módulo armasse e tratasse o sinal ***SIGINT***, no caso de o paciente não querer ficar à espera, possa cancelar a consulta através do ***<CTRL+C>***. Para isso foi criada a função ***handler()***, que por sua vez é chamada na ***main*** através do ***signal(SIGINT, handler)***, que tem como tarefa, enviar uma mensagem com o status da consulta a 5 para a ***mailbox*** com o número do PID da ***message queue***. Termina a função com o comando do print para avisar o utilizador de que o paciente cancelou o pedido e saindo do processo.

# *Servidor.c*

Comecei o desenvolvimento do módulo do servidor por criar uma função ***checkSharedMemory()*** que verificasse a existência da ***Shared Memory*** que irá armazenar a lista de consultas e os diferentes contadores com os diferentes tipos de consultas e o das consultas perdidas. Primeiro defini um valor inteiro com a flag ***IPC_EXCL***, em que verificamos que retorna -1 caso a ***Shared Memory*** com a nossa ***IPC_KEY*** já exista. Em seguida defini o que fazer caso a mesma não existe, isto é, o valor inteiro definido anteriormente ser diferente de -1, e nesse caso, criar efetivamente a ***Shared Memory*** e realizar o ***attach*** à mesma através de um ***pointer*** criado por uma estrutura definida em ***defines.h*** que tem como atributos, tudo o que pretendemos escrever em memória. Em seguida inicializei todos as posições da lista de consultas a -1, faço o print de que a ***Shared Memory*** foi inicializada e em seguida inicializo todos os contadores a 0 como pedido no enunciado. Em seguida queremos que dizer à função o que fazer caso a ***Shared Memory*** já existir, que no caso será apenas dar ***attach*** à mesma através de um ***pointer*** para começar a escrever as coisas em memória.

Avançando no enunciado, crio agora uma função ***recieveMessage()***, que tem como objetivo ficar à escuta de mensagens da ***mailbox 1*** da ***message queue***, isto é, da chegada de consultas do módulo ***Cliente.*** Começo por criar a ***message queue***, uma vez que o ***Servidor*** é inicializado primeiro que o ***Cliente*** e só a seguir disso é que fica à escuta de mensagens, e usando a função ***msgrcv().*** Após isso realizei a separação de informação através de uma função dada em aula pelo professor, que é a função ***substring***, definida no projeto em ***defines.h***, que tem como objetivo armazenar num vetor de caracteres, informação através de um ***splitter*** de outro vetor de caracteres. Após a separação de

informação criei um objeto do tipo Consulta como variável global, armazenando a informação recebida da message queue para que possa ser sempre possível, se necessário, a estas variáveis. Realizo o print da chegada de consulta com os respetivos dados e invoco a função ***dedicatedServer(),*** que corresponde às alíneas seguintes do enunciado.

Para o servidor dedicado, criei a função ***dedicatedServer()*** que tem como objetivo criar um processo filho. Comecei por fazer a verificação de vagas na lista de consultas criada em memória través de uma função auxiliar. Caso não haja vagas imprime que a lista está cheia e manda uma mensagem de status 4 para a mailbox com o número do PID que será recebida pelo Cliente na função ***recieveMessageWithPid()*** e sai do processo. Caso tenha vagas, realizei um ciclo for onde irá por a nova consulta na primeira posição da lista de consultas que seja -1 e associei as informações da variável do tipo consulta às informações da posição do vetor. Em seguida incrementei os contadores dependendo do tipo de consulta que é adicionada à lista de consultas. Para além disso realizei o envio da mensagem com status 2 para a ***mailbox*** com o número do PID da ***message queue***, faço a espera da ***DURAÇÃO*** através de um ***sinal alarm***, procedendo ao seu tratamento, e faço ainda o envio da mensagem com status 3. Após todas essas funções tenho a função ***verifyIfClienteCanceled()***, que fica à escuta de mensagens da ***mailbox*** com o número do PID, e caso essa mensagem seja 5, isto é, receba uma mensagem com status 5, faz o print do cancelamento da consulta e torna vaga da sala onde estava inscrito o paciente que foi cancelado. Em seguida, termina o processo.

Enquanto o processo filho realiza tudo isto, o processo pai irá tratar da alínea extra do enunciado, que basicamente consiste em tratar processos zombie sem nunca terminar o processo, para que caso queiramos adicionar Clientes ao Servidor infinitamente, os processos zombies não se vão acumulando. Resolvi isto usando o procedimento ***waitpid()*** que tem por base esperar que um específico processo filho termine, usando a flag ***WNOHANG*** para que os processos filhos sejam verificados sem nunca suspender o ***Servidor.***

Por fim, no último ponto do módulo do ***Servidor***, pede-se que seja armado e tratado o sinal ***SIGINT***, terminando o processo e mostrando no ecrã as estatísticas dos vários tipos de consultas. Criei então uma função ***handler()*** que por sua vez é invocada na ***main*** através do ***signal(SIGINT, handler)*** que realizar os prints das estatísticas e termina o processo através de um ***exit(0).***

Ao longo de todo este módulo fiz recurso a semáforos para ***acautelar a exclusão no acesso às zonas críticas***. Por outras palavras, usei um semáforo inicialmente com

valor 1 que irá decrementar o seu valor antes de fazer recurso à ***Shared Memory***, bloqueando o processo, e após o uso/escrita da ***Shared Memory*** deixa de estar bloqueado, incrementando o valor do semáforo em 1.

## *defines.h*

Trata-se de um ficheiro auxiliar que possui todos os ***includes*** e ***estruturas*** faladas anteriormente em cada módulo.