**IPV – Instituto Politécnico de Viseu**

**ESTGV – Escola Superior de Tecnologia e Gestão de Viseu**

**Departamento de Informática**

****

**Relatório do Trabalho Pratico 6**

**Licenciatura em Engenharia Informática**

**Realizado em**

**Sistemas Operativos**

**por**

**André Peixoto 18738**

**Gabriel Raperger 18509**

**Adelino Soares 18737**

**David Almeida 18709**

**Orientadores**

**ESTGV: Carlos Quental**

**Viseu, 2021**

# **Introdução**

Este trabalho visa demonstrar a comunicação entre processos utilizando os mecanismos de comunicação e sincronização lecionados. Neste trabalho existe um servidor que vai responder a pedidos de diversos clientes utilizando os mecanismos IPC como filas de mensagens, memoria partilhada e semáforos. Faz uso ainda de sinais, ficheiros e foi escrito na linguagem C.

# **Descrição**

* 1. **Cliente**

O programa cliente permite realizar pedidos ao servidor, obter a resposta do servidor e ainda executar comandos durante a sua execução.

Primeiramente, ao executar o cliente ele tenta aceder a memoria partilhada criada pelo servidor. Caso não tenha sucesso significa que o servidor não se encontra em execução e o programa encerra de imediato.

Caso consiga aceder a memoria partilhada, significa que o servidor está pronto para receber pedidos e envia o sinal SIGUSR2 para o servidor indicando que o cliente iniciou no servidor e a partir deste momento o cliente pode realizar pedidos.

Em seguida, utilizamos um semáforo *mutex*, criado pelo servidor, para sincronizar controlar o acesso a memoria partilhada. Primeiro executa uma operação de espera, isto é, decrementa o valor do semáforo e caso fique menor que zero o processo fica bloqueado a espera de que outro processo o desbloqueie. Depois faz o acoplamento com a memoria partilhada e escreve na primeira posição livre o seu *pid*. Ao terminar ele desacopla a memoria e assinala o semáforo autorizando a execução de outro processo.

Para enviar pedidos, o cliente espera que o utilizador insira um código e em seguida cria uma mensagem cujo remetente é o *pid* do cliente e o destino é um. Para explicar o porque do destino ser um, nos definimos que o servidor iria ler apenas as mensagens que tivessem destino um, mas poderia ter qualquer outro valor diferente do *pid* dos processos.

Para ler a resposta do servidor, o processo cliente cria um processo filho que recebe todas as mensagens cujo destino seja o *pid* do processo pai e em seguida apresenta a resposta na tela.

A escrita de pedidos e a respetiva leitura da resposta necessita de ser sincronizada, caso contrário o programa pode ficar num ciclo infinito, e para isto o programa cria dois semáforos, um pedido e outro resposta, iniciados respetivamente com um e zero. Ao enviar uma mensagem, o processo cliente realiza uma operação de espera no semáforo pedido, colocando-o a zero e ao terminar realiza uma operação de assinalar no semáforo resposta significando que o processo filho está pronto para receber a resposta. A semelhança do pai, o filho tem o mesmo comportamento, esperando o semáforo resposta e assinalando o semáforo pedido.

Durante a execução do cliente, é possível executar comandos através do atalho CTRL+Z. O atalho está associado ao sinal SIGTSTP, que ao ser pressionado executa uma rotina.

Por fim, o sinal SIGINT permite encerrar o programa. Ao ser enviado ele executa uma rotina de encerramento que remove os semáforos utilizados na sincronização dos processos e envia o sinal SIGUSR1 para o servidor indicando que aquele cliente terminou.

* 1. **Servidor**

O programa servidor é responsável por responder aos pedidos dos clientes, procurando na base de dados e enviando a resposta.

Primeiramente, ao iniciar o servidor ele verifica se a base de dados já esta criada. Caso não esteja ele permite criar uma nova com até 20 registos.

Em seguida ele cria todas os mecanismos IPC necessários, desde filas de mensagens, memoria partilhada e semáforos, associando os sinais a uma rotina.

Em seguida entra num ciclo no qual fica à espera de pedidos dos clientes que contenho destino igual a um. Ao receber um pedido ele faz uma pesquisa na base de dados, lendo o ficheiro até ao registo solicitado e devolvendo a linha. Em seguida envia uma resposta que se destina ao cliente que fez o pedido.

Na maior parte da sua execução, o servidor utiliza apenas filas de mensagens. Porem ao encerrar, o servidor executa uma rotina semelhante ao cliente em que, utilizando semáforos, acede a memoria partilhada em exclusão mútua e lê todos os *pid* que foram escritos pelos clientes e envia o sinal SIGUSR1 para todos.

Por fim, o servidor remove todos os mecanismos utilizados e encerra.

# **Alguns Problemas**

O nosso programa apresenta algumas falhas, as quais vamos descrever aqui.

Um problema simples é que quando um cliente introduz um código e realiza um pedido, o programa deveria aceitar apenas números, porem se escrevermos caracteres ele aceita de igual modo e inclusive recebe uma resposta do servidor que por vezes é o primeiro registo da base de dados.

Este problema poderia ser resolvido alterando o método de leitura de pedidos ou realizando algumas verificações no pedido introduzido.

Um problema mais grave é o facto de o servidor fazer uso de apenas um processo para responder aos pedidos, ou seja, não funciona de forma (pseudo)paralela. Isto representa um problema caso dois clientes façam um pedido em simultâneo.

Para resolver este problema, nos tentamos utilizar dois processos a semelhança do que fizemos no cliente. Porem este método representou um problema para enviar a resposta para o cliente uma vez que o cliente não conseguia obter a resposta do servidor.

# **Conclusão**

Por fim, é notável que o programa apresenta alguns problemas porem esta funcional.

A comunicação entre servidor e cliente é realizada, os mecanismos IPC utilizados são removidos, fizemos uso de mecanismos assíncronos como sinais e evitamos a ocorrência de processos zombie.

Todo o código está disponível nos documentos *cliente.c*, *servidor.c* e *header.h*.