Sistemas Operativos

Relatório - Simulação de um sistema de *home banking*

Estrutura das Mensagens

A **comunicação** entre o **servidor** e os seus **clientes** é feita recorrendo ao uso de estruturas fornecidas pelos docentes da unidade curricular, como a **tlv_request_t** (pedido) e a **tlv_reply_t** (resposta). Cada utilizador preenche uma *struct* do tipo tlv_request_t com a informação necessária para que o servidor possa processar o pedido. Esta struct é enviada através do FIFO **/tmp/secure_srv** e lida pelo servidor que, depois de processar a informação contida nela, constrói também um *struct* do tipo tlv_reply_t que contém informação sobre a resposta a ser entregue ao cliente respetivo, através de um FIFO **/tmp/secure_XXXXX** onde XXXXX representa o *process id* do cliente em questão.

O uso destas estruturas de dados permite que se realize a comunicação de forma **simples** e **eficiente** devido a várias razões. Por omissão, a definição de *structs* inere o uso de ***padding***, que faz com que a todos os membros da estrutura seja adicionado um "*gap"* para preencher o espaço de modo a que seja sempre ocupado o mesmo espaço por cada membro (que corresponde ao espaço ocupado pelo membro de maior tamanho). Desta forma a estrutura torna-se "alinhada" mas não é vantajoso em termos espaciais.

Para dar resposta ao problema anterior, as *structs* utilizadas são definidas com a extensão do compilador **[__attribute__((packed))]** que serve precisamente para impedir o *padding*. Desta forma, evita-se o envio de *bytes* desnecessários.

Para além da extensão utilizada, a comunicação é agilizada pelo facto de todas as mensagens seguirem o formato **TLV**(*type-length-value*). Este formato é bastante vantajoso para a leitura dos pedidos e respostas, na medida em que permite que apenas informação relevante da estrutura seja recolhida. Ou seja, inicialmente basta fazer a leitura do *type* e da *length* da mensagem e, mediante esses valores, lêem-se determinados campos dentro do *value*, evitando-se leituras de *bytes* em excesso.

Mecanismos de Sincronização

Para garantir o bom funcionamento do sistema, foi necessário recorrer a uma série de mecanismos de sincronização.

É imprescindível que o **acesso às contas** seja feito **exclusivamente** por cada *Bank Office* de modo a garantir a validade das operações sobre as mesmas. Por isso, para cada conta do *server* usa-se um ***mutex*** do tipo **pthread_mutex_t** que é *locked* antes de se realizar uma operação sobre a conta e *unlocked* depois. Este mecanismo é implementado através do uso das funções **pthread_mutex_init()**, **pthread_mutex_lock()**, **pthread_mutex_unlock()** da biblioteca <semaphore.h>.

Para o atendimento dos *requests* recorre-se à implementação de um *buffer* que simula uma **fila** de onde os *Bank Offices* (*threads*) vão ler esses pedidos. De modo a que este atendimento seja rápido e eficiente, é encarado como um problema de **produtor-consumidor** onde existem vários consumidores (*Bank Offices*). Assim sendo, usaram-se dois semáforos (**sem_t**) , da biblioteca <semaphore.h>, com o nome **notFull** e **notEmpty**. O primeiro corresponde ao número de pedidos que podem ser produzidos e o segundo ao número de pedidos que podem ser consumidos. Desta forma, os semáforos são usados para notificar o processo principal sobre o facto de poder ou não produzir um pedido (colocá-lo na fila) e para notificar os *Bank Offices* quando podem consumir (processar) esse mesmo pedido. Para este fim, o semáforo **notFull** é inicializado com o número de *Bank Offices* ativos - **sem_init(&notFull, 0, bankOfficesNo)**; e o **notEmpty** com o valor zero - **sem_init(&notFull, 0, 0)**.

Sempre que se quer **escrever** um pedido na fila é invocada a função **sem_wait(&notFull)**, escrevendo apenas quando há espaço na fila. Depois da escrita é invocada a função **sem_post(&notEmpty)** de modo a notificar o consumidor que pode ler mais um item.

Para se **ler** da fila de *requests* é primeiro invocada a função **sem_wait(&notEmpty)**, garantido que a leitura é feita quando existe pelo menos um item na fila. Após a leitura, é chamada a função **sem_post(&notFull)** notificando o produtor que existe espaço para produzir mais um pedido.

Encerramento do Servidor

Quando o administrador solicita o encerramento do servidor, o programa *server* invoca a função **chmod()** da biblioteca <sys/stat.h> de modo a impedir a escrita de mais mensagens no FIFO **/tmp/secure_srv**. Para além disso, dá-se o *set* de uma *flag* de modo a que a *thread* principal (*main*) não continue a tentar ler o FIFO, mas são processados aqueles que ainda estiverem pendentes, incluindo os que constarem no *buffer* desse FIFO.

Como os *Bank Offices* devem terminar de processar os *requests* que se encontram em fila de espera, estes finalizam o seu trabalho tendo em conta o tamanho da fila (quando for igual a zero) e a *flag* referida anteriormente.

O processo principal espera que todas as *threads* (*Bank Offices*) terminem, recolhendo-as através do uso da função **pthread_join()** da biblioteca <pthread.h>. Após todas as threads terminarem, a *main* toma conta da libertação da memória usada durante o processo, de fechar os ficheiros usados e da deleção do FIFO. De seguida encontra-se um diagrama de sequência que demonstra os pontos chave deste processo.