Faculdade de Engenharia da Universidade do Porto

Sistemas Operativos

Trabalho Prático 2 - Simulação de um sistema de home banking

Alexandre Miguel de Araújo Carqueja - up2017075049@fe.up.pt
Henrique José Santos - up201706898@fe.up.pt
Pedro Miguel Braga Barbosa Lopes Moás - up201705208@fe.up.pt

17 de maio de 2019

Estrutura das mensagens

As mensagens trocadas entre o user e o server foram as disponibilizadas no início do projeto no ficheiro types.h, especificamente **tlv_request** entre o utilizador e o servidor e **tlv_reply** entre o servidor e o utilizador.

Foi dado uso ao campo “length” de ambos para, consoante o tipo de request/reply, podermos ajustar a quantidade de informação enviada e como tal minimizar o custo de interação utilizador-servidor sem perdas de informação.

Mecanismos de sincronização

Utilizando várias threads no mesmo programa a aceder as mesmas variáveis, foi necessário adotar mecanismos de sincronização para que não haja alteração simultânea da mesma variável.

No processamento da fila de pedidos, resolveu-se como o problema do produtor/consumidor. Utilizaram-se dois semáforos, full e empty, inicializados a 0 e a N, respetivamente, sendo N o tamanho do buffer da fila, assim como um mutex, para que duas threads não acedam à fila ao mesmo tempo.

A main thread (Produtor) espera que a fila tenha espaço (empty > 0), colocando o pedido quando houver. Os bank offices, (Consumidores) esperam que a fila tenha algum conteúdo (full > 0), removendo um pedido quando tal existir.

Abaixo mostram-se excertos do código que realizam as tarefas de produtor e consumidor, respetivamente.

```
void addRequestToQueue(tlv_request_t request) {
   sem_wait(&queueEmptySem); // Wait
   pthread_mutex_lock(&queueMutex); // Lock
   queue_push(queue, request); // Append
   pthread_mutex_unlock(&queueMutex); // Unlock
   sem_post(&queueFullSem); // Signal
}
```

```
tlv_request_t getRequestFromQueue(int threadID) {
   sem_wait(&queueFullSem); // Wait
   pthread_mutex_lock(&queueMutex); // Lock
   tlv_request_t request = queue_pop(queue); // Take
   pthread_mutex_unlock(&queueMutex); // Unlock;
   sem_post(&queueEmptySem); // Signal
   return request;
}
```

Para a sincronização do acesso às contas, foi necessária outra abordagem. Foi decidido que, em primeiro lugar, deveria ser permitido aceder a duas contas diferentes ao mesmo tempo, caso contrário uma única operação poderia atrasar todas as outras, logo criou-se um mutex para cada conta existente, e funções lockAccount(id)/unlockAccount(id) que bloqueiam e desbloqueiam os respetivos mutexes.

Assim, sempre que se acede a uma conta, o programa deverá bloqueá-la antes de a alterar, e desbloqueá-la depois de realizar a operação, como por exemplo, validação, criação, ou consulta de saldo, como no exemplo abaixo.

```
(...)

 lockAccount(request.value.header.account_id, threadID);
 reply = handleBalanceRequest(request, threadID);
 unlockAccount(request.value.header.account_id, threadID);

(...)
```

No entanto, a realização de uma transferência implica que duas contas sejam bloqueadas. Isto poderá ser um problema, pois se uma thread pedir o acesso à conta A e esperar pela conta B, ao mesmo tempo que outra thread pede o acesso à conta B e espera pela conta A, a ocorrência de deadlock será bastante provável, não sendo possivel continuar a execução.

Para isso, fizeram-se funções lockDoubleAccount(id1, id2) e unlockDoubleAccount(id1, id2), que simplesmente bloqueia primeiro a conta com id menor, e desbloqueia primeiro a conta com id maior, respetivamente. Assim a espera circular é evitada e o deadlock não poderá ocorrer.

Encerramento do Servidor

Após o pedido de Shutdown do Servidor ser recebido, o primeiro passo realizado é a verificação de que este pedido foi feito pelo administrador do servidor. Caso isto não se verifique é enviada uma resposta de Erro do tipo OP_NALLOW.

Após a confirmação de que foi o administrador a realizar o pedido, começa-se por bloquear a escrita de mais pedidos ao servidor, isto é feito mudando as permissões da FIFO **/tmp/secure_srv** para ReadOnly (`fchmod(serverFifoFD, 0444)`).

O servidor continua depois o seu funcionamento normal, lendo o resto da FIFO até esta ficar vazia. Uma vez vazia iniciamos a eliminação dos Balcões virtuais (BankOffices).

Este processo começa apenas quando todos os pedidos já foram recebidos e completamente tratados por um Bank Office (Ou seja, quando a fila de pedidos se encontra vazia e o número de Bank Offices ativos é igual a 0).

```
bool queueEmpty = false;
while(!queueEmpty) {
	pthread_mutex_lock(&queueMutex); // Lock
	queueEmpty = queue_empty(queue);
	pthread_mutex_unlock(&queueMutex); // Unlock
}
while (getNumActiveOffices() > 0);
```

Seguidamente destruímos a fila de pedidos e também todas as contas criadas no servidor.