Faculdade de Engenharia da Universidade do Porto

Mestrado Integrado em Engenharia Informática e Computação

Sistemas Distribuídos

Grupo t5g04

Bruno Piedade – up201505668

Danny Soares – up201505509

Descrição do design e implementação de concorrência

Para gerir a concorrência no nosso projeto, recorremos a uma implementação baseada em *threads* e mutexes, por forma a assegurar que é possível executar múltiplos pedidos sem que haja nenhum problema de concorrência.

Para permitir a execução concorrente de múltiplos pedidos, foi usada uma *pool*, da classe *ThreadPoolExecutor,* que gere a execução das diferentes *threads*. Para cada pedido recebido é instanciado um objeto *Runnable* que processa a tarefa e, no caso dos protocolos BACKUP e RESTORE, é guardada uma referência para o mesmo num *Map* (classe *ConcurrentHashMap*), para permitir posteriores notificações aquando da receção de mensagens relevantes para o pedido que está a ser processado. Para garantir que os diferentes pedidos a ocorrer ao mesmo tempo não acedem ao mesmo objeto ao mesmo tempo, utilizámos *mutexes* da classe *ReentrantLock*, que fazem a gestão do acesso das múltiplas *threads* ao mesmo objeto. No código seguinte, encontra-se um exemplo da utilização desses *mutexes* no nosso trabalho, em que a variável *lock* é uma variável do tipo *ReentrantLock* e verifica que não há outra *thread* a aceder à variável *chunkPeers* no momento em que a *thread* a executar este código acede.

```
try{
        this.lock.lock();
        this.chunkPeers.add(new ArrayList<Integer>());
}
finally{
        this.lock.unlock();
}
```

O envio de mensagens é feito através de três *sockets multicast,* partilhados por todos os servidores, cujos acessos competitivos são geridos pela classe *TwinMulticastSocket*, recorrendo também a *locks,* para evitar que várias *threads* escrevam no mesmo *socket* ao mesmo tempo. Estes *sockets* estão associados a três *threads,* presentes nos servidores, que escutam constantemente os canais *multicast* e criam uma nova *thread* para processar qualquer mensagem recebida. Para o MC(*multicast control channel*) é criada uma *thread* da classe *ControlProtocol* que processa as mensagens do tipo STORED, GETCHUNK, DELETED e REMOVED; para o MDB(*multicast data backup channel*) uma *thread* da classe *StoreChunk* que processa as

mensagens do tipo STORECHUNK; para o MDR(*multicast data restore channel*) uma *thread* da classe *Chunk* que processa as menagens do tipo GETCHUNK. Estas *threads* realizam o *parsing* do conteúdo da mensagem, processam as mesmas e, caso seja necessário, enviam a reposta correspondente.

Para gerir os acessos concorrentes à memória que representa os ficheiros dos servidores, desenvolvimenos a classe *FileManager*, que guarda os ficheiros e os *chunks* que o seu servidor e funciona com o mecanismo *synchronized*, que garante que não há duas *threads* a aceder ao mesmo ficheiro ao mesmo tempo.

Concluindo, o nosso design de concorrência, baseado em *threads* e *mutexes*, permite que o sistema funcione corretamente com vários pedidos a ocorrer ao mesmo tempo.