Sistemas Distribuídos – COS470

2020/1

Trabalho Prático 3

1 Objetivo

O objetivo deste trabalho é projetar e implementar o algoritmo centralizado de exclusão mútua distribuída. A implementação pode ser em C, C++, Java, ou Go, esta última uma linguagem de programação aberta projetada para sistemas distribuídos e utilizada pela Google.

Além da implementação, você deve testar e avaliar seu programa. Você deve preparar um relatório, com no máximo 6 páginas, com as decisões de projeto e implementação das funcionalidades especificadas, assim como a avaliação dos estudos de caso. O relatório teve ter uma URL para seu código. Por fim, você deve preparar uma apresentação de no máximo 5 minutos sobre seu trabalho (como no relatório), que deve ser gravada e submetida. O trabalho pode ser feito individualmente ou em dupla.

2 Mensagens

Seu programa deve três tipos de mensagens:

1. `REQUEST`: Mensagem enviada por um processo para solicitar acesso à região crítica.
2. `GRANT`: Mensagem enviada pelo coordenador dando acesso à região crítica.
3. `RELEASE`: Mensagem enviada por um processo ao sair da região crítica.

Para facilitar, todas as mensagens devem ser strings com tamanho fixo dado por F bytes e separadores. Os separadores servem para indicar os campos dentro da mensagem, tais como o código de mensagem, identificador do processo, e o final da mensagem. Por exemplo, se $F = 10$ bytes, a mensagem `REQUEST` poderia ter o seguinte formado `1|3|000000`, o que neste caso indica que o separador é o caractere `|` e a mensagem `REQUEST` tem código 1 (cada mensagem deve ter um código), que o processo que gerou a mensagem tem identificador 3. Repare que os zeros no final servem apenas para garantir que a mensagem sempre tenha F bytes.

3 Coordenador

Você deve implementar o processo coordenador com três threads: uma apenas para receber a conexão de um processo, uma executando o algoritmo de exclusão mútua distribuída, e a outra atendendo a interface. Você deve usar uma estrutura de dados em fila, para armazenar os pedidos de acesso a região crítica. Você deve também contabilizar quantas vezes um processo foi atendido (recebeu a mensagem `GRANT`). A comunicação com os processos deve ser feita utilizando sockets (ou alguma outra API para comunicação). Repare que cada processo vai ter seu próprio socket, e você vai precisar manter uma estrutura de dados com esses sockets (para ler e escrever para os devidos sockets).

A thread de interface deve ficar bloqueada, lendo comandos do terminal, e processar os seguintes comandos: 1) imprimir a fila de pedidos atual, 2) imprimir quantas vezes cada processo foi atendido, 3) encerrar a execução. Cuidado: as duas threads vão acessar a mesma fila, e você deve sincronizar o acesso.

4 Processo

O processo deve se conectar ao coordenador via sockets (ou alguma outra API para comunicação). O processo possui apenas um socket, e deve conhecer o endereço IP e porta do coordenador (para conectar o socket).

Cada processo deve fica em um loop fazendo requisições de acesso a região crítica ao coordenador. Ao obter acesso, o processo abre o arquivo *resultado.txt* para escrita em modo *append*, obter a hora atual do sistema, escrever o seu identificador e a hora atual (incluindo milisegundos) no final do arquivo, aguardar k segundos (usando a função *sleep()*), e depois fechar o arquivo. Isto finaliza a região crítica. Este procedimento deve ser repetido r vezes, após o qual o processo termina.

Diferentes processos devem executar na mesma máquina e escrever no mesmo arquivo *resultado.txt*. O número de processos é dado por n.

5 Medição e avaliação

Uma execução com n processos e r repetições deve dar origem a um arquivo *resultado.txt* com nr linhas. Você deve verificar se o arquivo tem esse número de linhas e se as linhas fazem sentido (respeitam a ordem de evolução do relógio do sistema).

Use a interface do coordenador para inspecionar a situação da fila de requisições. Use $k = 3$ para ver a fila de requisições no coordenador crescer e depois diminuir.

Use os valores de tempo nas linhas do arquivo *resultado.txt* para determinar o tempo necessário para gerar o arquivo por completo. Use a diferença de tempo entre a última e primeira linhas do arquivo.

1. Faça um teste para $n = 2$, $r = 10$, e $k = 1$ para garantir que está tudo funcionando.

2. Teste 1 de escalabilidade do seu sistema. Fixar $r = 100$ e $k = 1$ e aumentar n, tal que $n \in \{2, 4, 8, 16, 32, 64, 128\}$. Para cada cenário, verifique se o arquivo *resultado.txt* foi gerado corretamente. Trace um gráfico com o tempo necessário para gerar o arquivo por completo.

3. Teste 2 de escalabilidade do seu sistema. Fixar $r = 1000$ e $k = 0$ e aumentar n, tal que $n \in \{2, 4, 8, 16, 32, 64, 128\}$. Para cada cenário, verifique se o arquivo *resultado.txt* foi gerado corretamente. Trace um gráfico com o tempo necessário para gerar o arquivo por completo.

Faça uma discussão dos resultados obtidos nos testes de escalabilidade.