Sistemas Distribuídos – COS470

2021/1

Trabalho Prático 3

1 Objetivo

O objetivo deste trabalho é projetar e implementar o algoritmo centralizado de exclusão mútua distribuída. Além da implementação, você deve testar e avaliar seu programa em diferentes cenários. 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, disponibilizada na nuvem, e a URL compartilhada. O trabalho deve ser realizado 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 identificador da 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 identificador 1 (cada mensagem deve ter um identificador), 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

O processo coordenador deve ser *multi-threaded*. Uma alternativa é ter uma thread apenas para receber a conexão de um novo processo, uma thread executando o algoritmo de exclusão mútua distribuída, e a outra atendendo a interface (terminal). Você deve usar uma estrutura de dados em fila, para armazenar os pedidos de acesso a região crítica. O coordenador deve gerar um log com todas as mensagens recebidas e enviadas (incluindo o instante da mensagem, o tipo de mensagem, e o processo origem ou destino).

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 o coordenador deve manter uma estrutura de dados com esses sockets (para ler e escrever para os devidos sockets, um para cada processo). Uma alternativa é usar um socket do tipo UDP, que não requer conexão (neste caso, o coordenador precisa manter apenas um socket para todos os processos).

A thread de interface deve ficar bloqueada, aguardando 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, fechar o arquivo, e depois aguardar k segundos (usando a função *sleep()*). 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

Os processos devem ser todos iniciados sequencialmente (sem retardo), por um script ou um outro processo. 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 estão corretas (respeitam a ordem de evolução do relógio do sistema, e se cada processo escreveu r vezes). Faça um programa avaliar se o arquivo *resultado.txt* gerado está correto.

Avalie o log gerado pelo coordenador para ver se o mesmo está correto. Por exemplo, as mensagens GRANT e RELEASE sempre ocorrem de forma intercalada (depois de um GRANT sempre há um RELEASE); a ordem dos processos das mensagens REQUEST deve ser a mesma que a ordem dos processos da mensagem RELEASE. Faça um programa avaliar se o arquivo de log gerado pelo coordenador está correto.

Para cada cenário abaixo, verifique se o arquivo *resultado.txt* e arquivo de log do coordenador foram gerados corretamente. Trace um gráfico em função de n com o tempo necessário para gerar o arquivo por completo (use a diferença de tempo entre a última e primeira linha do arquivo).

1. Teste 0 de funcionamento. Fixar $n = 2$, $r = 10$, e $k = 2$ para garantir que está tudo minimamente funcionando.

2. Teste 1 de escalabilidade do sistema. Fixar $r = 10$ e $k = 2$ e aumentar n, tal que $n \in \{2, 4, 8, 16, 32\}$.

3. Teste 2 de escalabilidade do seu sistema. Fixar $r = 5$ e $k = 1$ e aumentar n, tal que $n \in \{2, 4, 8, 16, 32, 64\}$.

4. Teste 3 de escalabilidade do seu sistema. Fixar $r = 3$ e $k = 0$ e aumentar n, tal que $n \in \{2, 4, 8, 16, 32, 64, 128\}$.

Faça uma discussão dos resultados obtidos nos testes de escalabilidade. Informe os possíveis problemas encontrados nos diferentes cenários.