Sistemas Distribuídos – COS470

2016/1

Trabalho Prático 3

1 Objetivos

O objetivo deste trabalho é se familiarizar com RPC (*Remote Procedure Call*) e algoritmos de exclusão mútua entre *processos*, ambos aspectos fundamentais no projeto e desenvolvimento de sistemas distribuídos. Para cada parte do trabalho, você deve desenvolver um programa na linguagem de programação de sua preferência mas que tenha suporte a RPC e mecanismos de sincronização, como semáforos. A sugestão é utilizar C ou C++, tendo em vista a proximidade das bibliotecas destas linguagens com o Sistema Operacional, oferecendo ao desenvolvedor (você) um maior controle.

Além da implementação, você deve testar seu programa executando os estudos de casos. Você deve preparar um relatório, com no máximo 4 páginas, com as decisões de projeto e implementação das funcionalidades especificadas, assim como a avaliação dos estudos de caso. Por fim, você deve preparar uma apresentação de no máximo 10 minutos sobre seu trabalho (como no relatório), a ser realizada no horário da aula. O trabalho pode ser individual ou em dupla.

2 Operações aritméticas em vetores com RPC

Considere o problema de aplicar uma função matemática sobre os elementos de um vetor. Por exemplo, calcular o logaritmo de cada elemento do vetor. Considere ainda que o processamento será feito *in place*, ou seja que o resultado da função matemática irá substituir o valor original no vetor. Projete e implemente um serviço utilizando RPC que ofereça esta funcionalidade. Você deve implementar ao menos três funções matemáticas, com ao menos uma delas sendo parametrizada. Por exemplo, a função potência onde o parâmetro é o valor da potência ao qual cada elemento do vetor será elevado: *int potencia_rpc(double a, double *vec, int vec_size)* eleva cada elemento do vetor *vec* que possui tamanho *vec_size* a potência *a*.

Projete e implemente um programa *multithreaded* que vai utilizar o serviço oferecido via RPC. Seu programa deve alocar um vetor de N posições e inicializá-lo com valores aleatórios. Para agilizar o processamento, o programa deve utilizar K threads que farão uso do serviço RPC simultaneamente. Uma forma de dividir o problema é instruir cada thread para trabalhar com N/K posições contíguas do vetor, cada uma fazendo apenas uma chamada RPC. Instrumente o tempo necessário para aplicar a função matemática no vetor (sem contar o tempo de inicialização).

Para o estudo de caso, considere $N = 10^8$ e $K = 1, 2, 4, 8, 16, 32, 64, 128$, e utilize as diferentes funções que foram implementadas no serviço. Para cada valor de K e função matemática, obtenha o tempo de execução rodando o programa 10 vezes para calcular o tempo médio de execução (para cada combinação de valores, teremos um tempo médio de execução). Apresente um gráfico mostrando o tempo médio de execução em função do número de threads para cada valor de K (cada função matemática deve ser uma curva no gráfico). Discuta os resultados obtidos.

3 Exclusão mútua entre processos

Considere o problema de garantir exclusão mútua a uma região crítica entre processos. Projete e implemente o algoritmo centralizado visto em aula, implementando as primitivas *request*, *grant* e *release*. Para a troca de mensagens entre processos, você pode utilizar *sockets* (como no

primeiro trabalho) ou qualquer outra primitiva de mais alto nível, como RPC. Instrumente o processo coordenador para logar os três tipos de mensagens com o tempo do relógio local (RTC) de cada mensagem, gerando um arquivo.

Para o estudo de caso, considere que K processos clientes tem acesso ao mesmo sistema de arquivos (por exemplo, executam na mesma máquina). Todos os processos clientes desejam escrever no mesmo arquivo, operação esta que precisa ser realizada exclusivamente para o bom funcionamento do sistema. Cada processo cliente dorme por um tempo aleatório entre $[0, T]$ segundos e ao acordar adiciona uma frase a um arquivo compartilhado entre todos os processos. A região crítica envolve abrir o arquivo, escrever uma frase[1] no final do arquivo, e fechar o arquivo. O processo deve repetir este procedimento e escrever X frases no arquivo, para depois concluir sua execução.

Considere duas formas dos processos entrarem no sistema distribuído: (i) *bulk arrival*, todos os processos iniciam execução simultaneamente; (ii) *sequential arrival*, um processo entra por vez no sistema, com o intervalo de tempo de Y segundos entre eles. Implemente estas duas formas dos processos entrarem no sistema.

Para o estudo de caso, considere $T = 1$, $K = 1, 2, 4, 8, 16, 32, 64, 128$, $X = 100$, e $Y = 1$. Teste seu programa para as duas formas de entrada de processos. Faça um programa que verifica se o arquivo gerado pelos processos está correto, ou seja, se não há inconsistências nas frases escritas no arquivo. Por fim, calcule o tempo para execução do sistema, ou seja, o intervalo de tempo desde a chegada do primeiro processo até a saída do último processo. Faça um gráfico do tempo de execução em função de K, executando 10 vezes cada cenário e reportando a média amostral do tempo de execução. O gráfico deve ter duas curvas, correspondendo as duas formas de entrada no sistema.

[1] Cada processo pode ter um conjunto de frases pré-definidas que são escritas uma depois da outra, circulando entre elas. Mas você pode pensar em outro mecanismo para gerar as frases.