MAC 438 - Programação Concorrente - Primeiro Semestre de 2001

Quarto Exercício-Programa: `ServContasTP`

Data de Entrega: 2 de julho de 2001

Este exercício deve ser desenvolvido em equipes de duas pessoas, a fim de suscitar discussão. Dúvidas sobre o enunciado devem ser enviadas para `reverbel-mac438@ime.usp.br`.

1 Preliminares

No exercício-programa anterior você implementou o programa `ServContas`, um servidor multithreaded que cria uma thread para tratar cada sessão com um cliente. Criar uma thread para cada sessão é um dos possíveis modelos de programação de servidores multithreaded. Os três modelos mais freqüentemente usados são:

Thread-per-request: Este modelo de programação cria uma nova thread para cada requisição de serviço proveniente de algum cliente. A thread efetua o serviço requisitado, manda uma mensagem de resposta para o cliente e encerra sua execução. O modelo thread-per-request é útil para servidores que tratam requisições de longa duração (como consultas a um banco de dados) emitidas por múltiplos clientes. Ele é menos útil no caso de requisições de curta duração, devido ao overhead de criação de uma thread a cada requisição. Pode também consumir muitos recursos do S.O. caso muitos clientes façam requisições simultâneas.

Thread-per-session: Esta variação do modelo thread-per-request amortiza por várias requisições o custo de criação de uma thread. Para cada cliente que se conecta com o servidor é criada uma nova thread, que tem a duração da sessão do cliente. O modelo thread-per-session é útil no caso de múltiplos clientes que travam conversações demoradas com o servidor. Não é útil se os clientes fazem só uma requisição, pois recai no modelo thread-per-request.

Thread pool: Esta variação dos modelos anteriores elimina o custo de criação de threads empregando um pool de worker threads criadas na inicialização do servidor. É útil para servidores que querem colocar um limite na quantidade de recursos do S.O. que eles consomem. A alocação de threads do pool pode ser feita por requisição ou por sessão. No primeiro caso, o tamanho do pool determina o número de requisições que podem ser processadas concorrentemente. As requisições concorrentes que ultrapassarem este número devem ser enfileiradas para posterior tratamento. No segundo caso, o tamanho do pool determina o número de sessões simultâneas com clientes. Os clientes que ultrapassarem este número devem ser enfileirados para posterior atendimento.

2 O problema

Neste exercício implementará uma versão do `ServContas` que usa o modelo thread pool. O novo programa, denominado `ServContasTP`, alocará uma thread do pool para tratar cada sessão com um cliente. Espera-se que o `ServContasTP` reaproveite praticamente todo o código do `ServContas`.

As worker threads do pool devem ser criadas na inicialização do servidor. Além destas, seu programa deve ter uma listener thread, que fica aguardando novas conexões de clientes (a listener thread pode ser a própria main). A cada conexão, a listener thread aloca uma worker thread do pool (caso haja alguma) para tratar a nova sessão. Quando a sessão se encerrar, essa worker thread voltará para o pool. Caso todas as worker threads estejam alocadas para tratar outras sessões, a listener thread enfileirará a nova sessão para posterior tratamento por uma worker thread que retorne ao pool.

O número de worker threads e o comprimento máximo da fila de sessões que estão aguardando uma worker thread são argumentos especificados na ativação do `ServContas`. Novas conexões de clientes devem ser recusadas caso a fila de sessões tenha comprimento máximo. A fila de sessões deve ser implementada como um bounded buffer com múltiplos consumidores (as worker threads) e um produtor (a listener thread).

3 Requisitos da solução

Os requisitos são os mesmos do `ServContas`, trocando-se o modelo de programação thread-per-session pelo modelo thread pool com alocação de uma thread do pool para cada sessão. Duas opções adicionais devem ser suportadas:

- `-workers` n, que especifica o número de worker threads;
- `-bound` n, que especifica o comprimento máximo da fila de sessões.

Além dos clientes do exercício programa anterior (os quais devem continuar funcionando com o `ServContasTP`), você deve entregar mais um cliente (ou conjunto de clientes) que compare os desempenhos dos dois servidores. Escreva esse(s) cliente(s) com o objetivo de tentar evidenciar a diferença de desempenho entre o `ServContas` e o `ServContasTP`, na situação em que essa diferença deveria ser maior: sessões bem curtas, com uma só requisição por sessão. Ou seja, faça o cliente executar um laço que abre uma sessão com o servidor, envia uma requisição, aguarda a resposta e fecha a sessão. O `ServContas` criará uma nova thread a cada volta, enquanto que o `ServContasTP` só alocará uma worker thread do pool.

Use como medidas de desempenho o tempo de resposta para um cliente e a vazão (throughput) com múltiplos clientes.

- O tempo de resposta é o tempo de execução, por um cliente, de uma volta de um laço "abre sessão, manda requisição, aguarda resposta e fecha sessão".
- A vazão é número de sessões que o servidor consegue tratar por unidade de tempo quando múltiplos clientes rodam laços como esse.

4 Seu arsenal

Seu arsenal é o mesmo do exercício programa anterior. Só uma observação aqui: para comparar os desempenhos do `ServContas` e do `ServContasTP` é interessante rodar ambos os servidores numa JVM que use threads do S.O. (native threads) em vez de "green threads". Acredito que o overhead de criação de threads seja maior com native threads. Se isso for verdade, a diferença de desempenho entre os servidores aparecerá mais. As três versões de JDK instaladas na rede Linux (veja `http://www.linux.ime.usp.br/java/`) tem native threads.

5 Sobre a entrega

Você deverá entregar três coisas:

- um arquivo `tar.gz` contendo sua solução (arquivos-fonte, makefile, README, ...);
- uma listagem impressa dos seus arquivos-fonte;
- um relatório comparando os desempenhos do `ServContas` e do `ServContasTP`.

O arquivo `tar.gz` deve incluir os fontes do `ServContas`, do `ServContasTP` e dos clientes. O `README` deve incuir uma explicação de como rodar cada cliente.

Por favor, entregue o relatório e a listagem **na secretaria do MAC**, em um saco plástico devidamente fechado, contendo também um disquete com o arquvo tar.gz.

Trabalhos atrasados não serão aceitos!

Bom trabalho!