Universidade Federal de Santa Catarina – UFSC
Departamento em Informática e Estatística
INE 5645 – Programação Paralela e Distribuída
Semestre 2024/2

Definição do Trabalho 1: Programação Paralela

Este trabalho visa explorar o uso de padrões para programação *multithread*. Cada grupo irá explorar (pelo menos) os modelos de programação *produtor/consumidor* e *pool de threads* na solução do problema.

Descrição do problema: Implementação de um servidor de contas com atendimento baseado em *pool de threads*

Este trabalho consiste em desenvolver um *servidor multithreaded*. A ideia geral é que *clientes* possam gerar requisições sobre contas bancárias, que serão atendidas por um servidor. Para aumentar a vazão do serviço e tirar proveito de arquiteturas com múltiplos núcleos de processamento, o servidor delega o processamento das requisições dos clientes a *threads trabalhadoras*.

A seguir são descritos os componentes do serviço bancário:

Contas bancárias: O serviço deve manter informação sobre contas de usuários. Cada conta de usuário tem um identificador da conta (um número inteiro positivo) e um saldo atual (um número real, que pode ser positivo ou negativo).
O conjunto de contas de todos os usuários pode ser armazenado em uma estrutura de dados da sua preferência (ex. uma tabela hash, uma lista, etc.). É importante apenas que as operações do serviço possam acessar contas de usuários e efetuar atualizações nas contas.

Operações: Neste trabalho, cada grupo deve disponibilizar códigos individuais para a implementação de 3 operações do serviço (cada operação pode ser uma função, por exemplo):

a) *Depósito em conta corrente*: Esta operação recebe um identificador de conta (um número inteiro positivo) e o valor de depósito (um número real, que pode ser positivo ou negativo). Note que esta operação pode ser executada tanto para depósitos quanto saques, dependendo se o valor de depósito é positivo ou negativo;
b) *Transferência entre contas*: Dadas duas contas bancárias, origem e destino, e um valor de transferência, esta operação deve debitar o valor de transferência da conta de origem e somar este valor na conta destino;
c) *Balanço geral*: Esta operação gera um balanço geral de todas as contas, imprimindo na tela cada conta e o seu respectivo valor no momento em que a operação foi inicializada. Note que o balanço geral apresenta uma “fotografia” instantânea do estado das contas.

Para cada operação, além de implementar a lógica específica da operação, insira uma função sleep (ou usleep) com um valor de espera definido para cada operação. Este tempo de espera simboliza um tempo de processamento da requisição. A inserção desta espera

artificialmente manterá a operação em execução por mais tempo, ajudando a ilustrar os entrelaçamentos na execução do sistema concorrente.

Servidor: O servidor tem a responsabilidade de receber requisições dos clientes, despachar essas requisições para que as operações sejam executadas e solicitar balanços gerais sobre as contas, periodicamente. Para garantir a eficiência do processamento, o servidor utiliza um *pool de threads* para lidar com múltiplas requisições simultaneamente.

O *servidor* é representado por uma *thread* e ele consome requisições de clientes de uma fila de requisições e atribui cada requisição a uma *thread trabalhadora* que esteja livre para atender a requisição. O servidor permanece em um laço de repetição sempre buscando por novas requisições na fila de requisições, bloqueando quando não houver requisições pendentes ou quando não houver *threads trabalhadoras* disponíveis. Além disso, a cada 10 operações de clientes, o servidor adiciona uma operação de balanço geral (c) na fila de requisições.

Threads trabalhadoras: O sistema é configurado para disponibilizar um *pool de threads* trabalhadoras. Cada *thread* tem um estado associado (livre ou em execução). As *threads* ficam aguardando um evento de ativação vindo do servidor. Este evento deve indicar a *thread* qual a operação que deve ser executada e os respectivos parâmetros. Ao ser notificada, a *thread* atualiza o seu estado para *em execução* e executa a operação conforme descrito anteriormente. Ao finalizar a execução, a *thread* volta para o estado *livre* e notifica o servidor.

Clientes: As *threads* cliente geram as operações (a) e (b), sorteando contas de clientes e valores para as operações aleatoriamente. Este comportamento é repetido constantemente ou até que algum critério de parada (a critério do grupo) seja alcançado. Entre a geração de uma requisição e outra, a thread cliente executa a função sleep. Quanto maior o sleep, menor será a taxa de chegada de requisições, e vice-versa.

Tenha atenção às condições de corrida no seu programa e escolha adequadamente estruturas de sincronização para facilitar a coordenação entre os participantes do seu programa e prevenir inconsistências nos resultados retornados pelos serviços.

Execução

Para simular a execução, você deve parametrizar o seu programa. Parâmetros podem ser lidos de arquivo ou dados como parâmetros de entrada na linha de comando. É importante que o grupo varie alguns parâmetros nas execuções para avaliar a influência dos mesmos na execução, por exemplo: tamanho do pool de threads, número de clientes e taxa de geração de novas requisições, tempo de serviço (o sleep para cada operação). O programa pode executar infinitamente ou pode ser estabelecido um critério para término (por exemplo, executar por X segundos ou terminar após o atendimento de *n* requisições).

O programa pode ser desenvolvido em qualquer linguagem de programação, mas visando a execução paralela das requisições sempre que possível (lembre que em Python *threads* não executam em paralelo). Ainda, podem ser utilizadas bibliotecas prontas que implementam estruturas de dados, aleatoriedade ou outras funções referentes à manipulação de dados. Porém, o *pool de threads* e o *modelo produtor/consumidor*, deverão

ser implementados pelo grupo. Também não é permitido o uso de estruturas de dados (pilhas, filas, listas, tabelas, etc.) que implementam exclusão mútua internamente, ou seja, o controle de concorrência deve ser implementado pelo grupo.

Entrega

O trabalho consiste em:

1. Implementar um programa que simule a execução do serviço seguindo os requisitos descritos acima;
2. Breve relatório que indique as principais decisões e estratégias de implementação utilizadas, instruções sobre como compilar e executar o código produzido, além de exemplos de saídas de execução com diferentes parametrizações. Ao final, discuta quais conclusões você observa ao variar parâmetros do sistema.

O trabalho pode ser realizado em **grupos de até 3 participantes**. O trabalho será apresentado em sala de aula.

O código-fonte e relatório devem ser enviados pelo Moodle para análise e avaliação.

Os nomes dos participantes do grupo devem constar no relatório entregue no Moodle. Participantes com nomes não referenciados não serão considerados membros do grupo.