UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE CIÊNCIAS EXATAS E DA TERRA
DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA
DIM0612 – PROGRAMAÇÃO CONCORRENTE | 2017.1

Trabalho prático

Programação Concorrente em Go

Objetivo

O objetivo deste trabalho é estimular o projeto, implementação e avaliação de soluções para problemas por meio de programação concorrente utilizando a linguagem de programação Go (https://golang.org/).

O problema

Um dos maiores desafios no mercado altamente competitivo de hoje é o que se denomina *Desafio do Fornecimento*. As empresas competem de maneira inovadora e agressiva para que seus produtos e serviços sejam entregues aos clientes no menor tempo e com o menor custo possível. Nesse cenário, o uso de soluções e ferramentas computacionais tem se mostrado um diferencial estratégico significativo. Mais e mais empresas estão investindo na modernização de seus sistemas de apoio para que se destaquem com relação aos seus concorrentes de mercado.

Uma das empresas diretamente envolvidas nessa disputa é a *ATL Transportes e Logística*. Essa empresa de transportes é responsável pela entrega dos produtos de uma série de grandes fornecedores e é historicamente muito conceituada em seu segmento de mercado. Contudo, com o advento das vendas pela Internet, a empresa tem percebido que sua capacidade de processamento de pedidos está gradativamente se mostrando aquém do necessário. Se continuar nesse ritmo de queda, em pouco tempo a ATL perderá o destaque no mercado alcançado ao longo dos anos.

Ao perceber a necessidade de mudar suas operações, a ATL decidiu que era hora de rever o processo de atendimento a pedidos. Atualmente a recepção de pedidos para entrega de produtos é feita por telefone e/ou pela sua página na Internet. Apesar de permitir que vários pedidos sejam feitos ao mesmo tempo em uma capacidade de 50 atendentes por telefone e 300 solicitações simultâneas pela Internet, o processamento interno dos pedidos é feito por um sistema de *software* que os analisa um de cada vez. Foi essa a restrição no fluxo de operações identificada pela equipe interna de TI da ATL.

Como todos os recursos humanos de TI da empresa estão alocados em outros projetos e dispõem de pouco ou nenhum tempo para a realização dessa modernização, o diretor de TI da ATL decidiu abrir uma concorrência no mercado para que diversas empresas apresentem uma solução para melhorar a capacidade de processamento de pedidos. A ideia geral é que seja implementada uma solução concorrente assíncrona para processamento dos pedidos seguindo os princípios gerais do *Problema dos*

Produtores e Consumidores, clássico em sistemas concorrentes.

Para essa concorrência aberta, a ATL não exige que seja necessário implementar um sistema de *software* completo, mas apenas um protótipo que torne possível evidenciar os resultados de uma futura implementação formal. Os pedidos serão enviados em um formato de dados que consiste de um identificador numérico e um pacote de dados em formato de texto. Os clientes farão uso da nova ferramenta para alimentar um *buffer* interno com capacidade para 5000 pedidos. Um processamento interno assíncrono consumirá os pedidos e os processará individualmente, de modo que os clientes não esperarão *on-line* pela confirmação, mas receberão uma resposta posterior.

Tarefas

A tarefa central a ser realizada neste trabalho é conceber e implementar na linguagem de programação Go o protótipo requerido pela ATL para processamento dos pedidos utilizando conceitos e técnicas de programação concorrente. A solução deverá ser desenvolvida em três etapas, cada uma resultando em uma versão da implementação da solução, que deverá ser conservada para fins de avaliação comparativa.

A implementação deverá garantir corretude do programa com relação a concorrência e aplicar de forma adequada as facilidades de gorotinas. Para promover sincronização entre as gorotinas, você poderá fazer uso de canais de comunicação (com ou sem *buffer*) e/ou de mecanismos de sincronização avançada providos pelo pacote `sync` da linguagem. Seu código fonte deverá estar **obrigatoriamente** comentado, a fim de prover entendimento adequado daquilo que foi realizado.

Além da implementação, deverá ser elaborado um relatório escrito descrevendo, pelo menos:

- como a solução foi projetada;
- a lógica de sincronização utilizada, em termos dos mecanismos empregados e como ela é feita entre os fluxos de execução do programa;
- como é garantida a corretude da solução com relação a concorrência;
- os resultados obtidos em experimentos computacionais realizados ao final da consecução de cada etapa;
- uma análise comparativa entre as diferentes versões da solução produzidas em cada etapa
- eventuais dificuldades encontradas.

Etapa 1

Crie agentes baseados em gorotinas que consumam um *buffer* de 5000 posições previamente preenchido com os pedidos (ou seja, apenas os agentes consumidores deverão ser implementados nesta etapa). O consumidor eliminará continuamente os pedidos do *buffer* e seu tempo de processamento deverá ser simulado por uma pausa de 500 milissegundos. Ao final de cada processamento, deverá ser exibido um *log* com a identificação do agente consumidor, a identificação do pedido, o horário de início e o horário de término do processamento. Quando o *buffer* estiver vazio, os agentes consumidores serão bloqueados. Por ora, não há a necessidade de se preocupar com a sincronização necessária entre os diversos agentes consumidores.

Uma vez concluída a implementação, faça um experimento no qual sejam iniciadas quantidades dis-

tintas de agentes consumidores simultâneos variando entre 1, 5, 10, 50, 100, 500 e 1000. Visando obter maior significância estatística, realize a execução de pelo menos dez execuções para cada uma dessas quantidades e armazene o tempo de execução de cada uma delas. Para exibição dos resultados, crie uma tabela na qual conste os tempos médio, mínimo e máximo, além do desvio padrão dentre as dez execuções para cada quantidade de agentes consumidores. Elabore também um gráfico mostrando os tempos médios de execução para cada quantidade de agentes consumidores.

Etapa 2

Altere a implementação realizada na etapa anterior de modo que o *buffer* seja agora preenchido por um agente (também baseado em gorotinas) produtor, de modo que haja sincronização entre os agentes produtores e consumidores. Os agentes produtores deverão criar um pedido e seu tempo de processamento deverá ser simulado por uma pausa de 500 milissegundos. Cada agente produtor alimentará o *buffer* continuamente enquanto houver espaço nele e deverá ser bloqueado quando estiver cheio. Quando houver um novo espaço disponível no *buffer* resultante do eventual consumo de um pedido por um agente consumidor, o agente produtor deverá ser reativado. Ao final de cada processamento, deverá ser exibido um *log* com a identificação do agente produtor ou consumidor, a identificação do pedido, o horário de início e o horário de término do processamento.

Uma vez concluída a implementação, faça um novo experimento no qual sejam iniciadas quantidades distintas tanto de agentes produtores quanto consumidores simultâneos variando entre 1, 5, 10, 50, 100, 500 e 1000. Visando obter maior significância estatística, realize a execução de pelo menos dez execuções para cada uma dessas quantidades e armazene o tempo de execução de cada uma delas. Para exibição dos resultados, crie uma tabela na qual conste os tempos médio, mínimo e máximo, além do desvio padrão dentre as dez execuções para cada quantidade de agentes produtores e consumidores. Elabore também um gráfico mostrando os tempos médios de execução para cada quantidade de agentes.

Dica: O critério de parada para os experimentos depende significativamente de que estrutura de dados será utilizada para representar o *buffer* e de como os agentes consumidores realizam sua execução sobre ele. Há pelo menos dois possíveis casos:

1. Se os consumidores estiverem literalmente retirando os itens do *buffer*, muito dificilmente os produtores conseguirão enchê-lo até as 5000 posições. Nesse caso, poderia haver uma espécie de contador que, quando tiver o valor 5000 alcançado, os produtores pararão de produzir novos itens. Como os consumidores estão removendo itens do *buffer*, eles serão suspensos quando o *buffer* estiver vazio.
2. Se o tamanho do buffer for mantido constante em 5000 (por exemplo, se o *buffer* tiver sido implementado como um *array* convencional), os produtores deverão ser suspensos quando o *buffer* estiver cheio. Nessa implementação, a ação de um consumidor seria marcar o item do *buffer* como consumido, de modo que os consumidores seriam suspensos quando não houverem mais itens a serem consumidos.

Etapa 3

Altere a implementação realizada na etapa anterior para atender a um novo requisito colocado pela diretoria de TI da ATL: agora deseja-se garantir que cada um dos pedidos seja processado na ordem em que foi solicitado, numa política FIFO (*first-in*, *first-out*). Você deverá aprimorar sua implementação

do agente consumidor de modo a garantir essa especificação.

Após a atividade de implementação, faça novamente o experimento realizado na etapa anterior, agora com o requisito de garantia de ordem do processamento dos pedidos. Para discussão, compare os resultados obtidos agora com os obtidos anteriormente e analise o impacto (se houver) apresentado pela inclusão desse novo, não só em termos de desempenho, mas também em termos de simplificação do código e facilidade de manutenção futura.

Autoria e política de colaboração

O trabalho poderá ser feito **individualmente ou em equipe composta por no máximo dois estudantes**, sendo que, neste último caso, é importante, dentro do possível, dividir as tarefas igualmente entre os integrantes da equipe. O trabalho em cooperação entre estudantes da turma é estimulado, sendo aceitável a discussão de ideias e estratégias. Contudo, tal interação não deve ser entendida como permissão para utilização de (parte de) código fonte de outras equipes, o que pode caracterizar situação de plágio. Trabalhos copiados em todo ou em parte de outras equipes ou da Internet serão sumariamente rejeitados e receberão nota zero.

A critério do professor, qualquer equipe pode ser eventualmente convocada para uma entrevista cujo objetivo é confirmar a autoria do trabalho desenvolvido e determinar a contribuição real de cada integrante. Durante a entrevista, cada membro da equipe deverá ser capaz de explicar, com desenvoltura, qualquer parte do trabalho, mesmo que esta tenha sido desenvolvida por outro membro da equipe. Portanto, é possível que ocorra, após a entrevista, redução da nota geral do trabalho ou ajustes nas notas individuais com vistas a refletir a verdadeira contribuição de cada membro da equipe.

Entrega

O trabalho deverá ser entregue até as **23h59 do dia 23 de junho de 2017, prazo que não será estendido**. Um membro de cada equipe deverá submeter, exclusivamente através da opção *Tarefas* da Turma Virtual do SIGAA, um único arquivo compactado contendo todos os códigos fonte referentes à implementação do trabalho, disponibilizados sem erros e devidamente testados e documentados, além do relatório escrito, preferencialmente em formato PDF. Se for o caso, é possível fornecer o endereço para um repositório destinado ao controle de versões, porém esta opção não exclui a necessidade de submissão dos arquivos via SIGAA.

Avaliação

A avaliação deste trabalho será feita principalmente sobre os seguintes critérios: (i) utilização correta dos conceitos e mecanismos de sincronização de gorotinas vistos na disciplina; (ii) a corretude da execução do programa implementado, tanto com relação a funcionalidades quanto a concorrência; (iii) a aplicação de boas práticas de programação, incluindo legibilidade, organização e documentação de código fonte, e; (iv) qualidade do relatório produzido. O trabalho possuirá nota máxima de 5,0 (cinco) pontos e será contabilizada para a terceira unidade da disciplina.