Universidade Federal da Paraíba
Centro de Informática / Departamento de Informática
Disciplina: Redes de Computadores I – 3ª Avaliação Parcial

Orientações:

- São 41 alunos, distribuídos em 10 equipes com, no máximo, 4 alunos cada (exceto 1 equipe que terá 5 alunos);
- Cada equipe deverá escolher entre a Atividade 1 (são 4 possibilidades) e a Atividade 2. Serão 8 equipes desenvolvendo a Atividade 1 e 2 equipes desenvolvendo a Atividade 2;
- A Atividade 2 está descrita no Capítulo 2 do livro Redes de Computadores e a Internet (Kurose & Ross) e o código disponibilizado está em anexo;
- Cada equipe deverá enviar email para giorgiamattos@gmail.com informando os integrantes da equipe e a atividade escolhida;
- O que deve ser entregue: os códigos fontes da implementação, relatório de como a atividade foi desenvolvida (dificuldades, facilidades, como a equipe resolveu/solucionou questões do projeto da atividade, e o que mais julgar necessário) e slides da apresentação, se houver, (70% da nota);
- Cada equipe deve apresentar, em 15 minutos no máximo, a sua atividade (30% da nota);
- As apresentações acontecerão nos dias 11 e 13/06 e cada equipe deve agendar a sua apresentação.

Atividade 1 – Projeto de Programação com Sockets (JOGO DA VELHA)

O projeto consiste em desenvolver um pequeno jogo com sockets UDP e TCP, usando a linguagem C, C++ ou Java. Os jogo é para 2 jogadores onde as jogadas vão se alternando entre os jogadores. Exemplo: jogo da velha, batalha naval, xadrez, damas.
Cada grupo fará uma combinação de jogo+protocolo+arquitetura+linguagem e deve implementar o programa de acordo com os requisitos solicitados. Também deve produzir um texto discutindo como as questões a seguir foram abordadas. Os fontes, executáveis e o texto devem ser enviados através do sigaa até a data 13/06/2018.

Arquiteturas:
• cliente-servidor, direto - sem intermediários.
• clientes-central, clientes interagem através de uma central.

Cada arquitetura tem aspectos as serem considerados:
• Central: onde vai ser executado?, como os clientes sabem onde está? Como registrar? Como sincronizar o jogo e as jogadas?
• Direto: como

Além do jogo, em si, o grupo terá que pensar em alguns pontos relevantes na aplicação:
• São dois jogadores que não se conhecem. Como um jogador obtém a referência do outro?
• Depois de obter a referência, como os dois jogadores tornam-se disponíveis para jogar, e combinam quem começa primeiro?
• Como as mensagens contendo as jogadas são representadas e manipuladas?
• Como terminar o jogo? Isto é: como terminar o jogo, decidir quem ganhou, notificar serem notificados e, finalmente liberar os recursos (socket, memória, etc.)?
• Um jogo pode ser interrompido no meio e depois reiniciado? E como ele recomeçará?
• O jogo possui requisitos de segurança contra trapaças? Por exemplo, um jogador consegue fazer duas jogadas seguidas antes do adversário fazer a sua?
• O que acontece se um jogador demorar muito na sua jogada?

Distribuição dos requisitos

Número	Grupo	Central	Direto	Java	C/C++	TCP	UDP	Jogo da Velha
1	Equipe 1	x		x		x		x
2	Equipe 2	x		x			x	x
3	Equipe 3		x		x	x		x
4	Equipe 4		x		x		x	x

Regras do Jogo da Velha

- O tabuleiro é uma matriz de três linhas por três colunas.
- Dois jogadores escolhem uma marcação cada um, geralmente um círculo (O) e um xis (X).
- Os jogadores jogam alternadamente, uma marcação por vez, numa lacuna que esteja vazia.
- O objetivo é conseguir três círculos ou três xis em linha, quer horizontal, vertical ou diagonal, e ao mesmo tempo, quando possível, impedir o adversário de ganhar na próxima jogada.
- Quando um jogador conquista o objetivo, costuma-se riscar os três símbolos.

Atividade 2: Projeto de Programação com Sockets (PROXY CACHE)

Neste laboratório, você desenvolverá um pequeno servidor Web proxy que também será capaz de fazer cache de páginas Web. Este será um servidor proxy bem simples, que apenas entenderá requisições GET simples, mas será capaz de manipular todos os tipos de objetos, não apenas páginas HTML, mas também imagens.

Código

O código está dividido em três classes:

- `ProxyCache` – compreende o código de inicialização do proxy e o código para tratar as requisições.
- `HttpRequest` – contém as rotinas para analisar e processar as mensagens que chegam do cliente.
- `HttpResponse` – encarregada de ler as respostas vindas do servidor e processá-las.

Seu trabalho será completar o proxy de modo que ele seja capaz de receber requisições, encaminhá-las, ler as respostas e retorná-las aos clientes. Você precisará completar a classes `ProxyCache`, `HttpRequest`, e `HttpResponse`. Os lugares onde você precisará preencher o código estarão marcados com /* Fill in - Preencher */. Cada lugar pode exigir uma ou mais linhas de código.

Nota: Conforme será explicado abaixo, o proxy utiliza DataOutputStreams para processar as respostas dos servidores. Isso ocorre porque as respostas são uma mistura de texto e dados binários, e a única cadeia de entrada em Java que permite trabalhar com ambos ao mesmo tempo é a DataOutputStream. Para obter o código a ser compilado, você deve usar o argumento –deprecation para o compilador, conforme segue:

```
javac –deprecation *.java
```

Se você não usar o flag –deprecation, o compilador irá se recusar a compilar seu código.

Executando o proxy

Para executar o proxy faça o seguinte:

```
java ProxyCache port
```

onde *port* é o número da porta que você quer que o proxy escute pela chegada de conexões dos clientes.

Configurando seu browser

Você também vai precisar configurar seu browser Web para usar o proxy. Isso depende do seu browser Web. No Internet Explorer, você pode ajustar o proxy em "Internet Options" na barra Connection em LAN settings. No Netscape (e browsers derivados, como Mozilla), você pode ajustar o proxy em Edit → Preferences e então selecionar Advanced e Proxies.

Em ambos os casos, você precisará informar o endereço do proxy e o número da porta que você atribuiu a ele quando foi inicializado. Você pode executar o proxy e o browser na mesma máquina sem nenhum problema.

Funcionalidades do proxy

O proxy funciona da seguinte forma:

1. O proxy escuta por requisições dos clientes.
2. Quando há uma requisição, o proxy gera um novo thread para tratar a requisição e cria um objeto HttpRequest que contém a requisição.
3. O novo thread envia a requisição para o servidor e lê a resposta do servidor dentro de um objeto HttpResponse.
4. O thread envia a resposta de volta ao cliente requisitante.

Sua tarefa é completar o código que manipula o processo acima. A maior parte dos erros de manipulação em proxy é muito simples, e o erro não é informado ao cliente. Quando ocorrem erros, o proxy simplesmente pára de processar a requisição e o cliente eventualmente recebe uma resposta por causa do esgotamento da temporização.

Alguns browsers também enviam uma requisição por vez, sem usar conexões paralelas. Especialmente em páginas com várias imagens, pode haver muita lentidão no carregamento delas.

Caching

Realizar o caching das respostas no proxy fica como um exercício opcional, pois isso demanda uma quantidade significativa de trabalho adicional. O funcionamento básico do caching é descrito a seguir:

1. Quando o proxy obtém uma requisição, ele verifica se o objeto requisitado está no cache; se estiver, ele retorna o objeto do cachê, sem contatar o servidor.
2. Se o objeto não estiver no cache, o proxy traz o objeto do servidor, retorna-o ao cliente e guarda uma cópia no cache para requisições futuras.

Na prática, o proxy precisa verificar se as respostas que possui no cache ainda são válidas e se são a resposta correta à requisição do cliente. Você pode ler mais sobre caching e como ele é tratado em HTTP na RFC-2068. Para este laboratório, isto é suficiente para implementar a simples política acima.

Dicas de programação

A maior parte do código que você precisará escrever está relacionada ao processamento de requisições e respostas HTTP, bem como o tratamento de sockets Java.

Um ponto notável é o processamento das respostas do servidor. Em uma resposta HTTP, os cabeçalhos são enviados como linhas ASCII, separadas por seqüências de caracteres CRLF. Os cabeçalhos são seguidos por uma linha vazia e pelo corpo da resposta, que pode ser dados binários no caso de imagens por exemplo.

O Java separa as cadeias de entrada conforme elas sejam texto ou binário. Isso apresenta um pequeno problema neste caso. Apenas DataInputstreams são capazes de tratar texto e binário simultaneamente; todas as outras cadeias são puro texto (exemplo: BufferedReader), ou puro binário (exemplo: BufferedInputStream), e misturá-los no mesmo socket geralmente não funciona.

O DataInputstream possui um pequeno defeito, pois ele não é capaz de garantir que o dado lido possa ser corretamente convertido para os caracteres corretos em todas as plataformas (função DataInputstream.readLine()). No caso deste laboratório, a conversão geralmente funciona, mas o compilador verá o método DataInputstream.readLine() como “deprecated” (obsoleto) e se recusará a compilá-lo sem o flag –deprecation.

É altamente recomendável que você utilize o DataInputStream para ler as respostas.

Exercícios Extras

Quando você terminar os exercícios básicos, tente resolver os seguintes exercícios opcionais.

1. Melhor tratamento de erros. No ponto atual, o proxy não possui nenhum tratamento de erro. Isso pode ser um problema especialmente quando o cliente requisita um objeto que não está disponível, desde que a resposta usual “404 Not Found” não contenha corpo de resposta e o proxy presuma que existe um corpo e tente lê-lo.
2. Suporte para o método POST. O proxy simples suporta apenas o método GET. Adicione o suporte para POST incluindo o corpo de requisição enviado na requisição POST.
3. Adicione caching. Adicione o funcionamento simples de caching descrito acima. Não é preciso implementar nenhuma política de substituição ou validação. Sua implementação deve ser capaz de escrever respostas ao disco (exemplo: o cache) e trazê-las do disco quando você obtiver um encontro no cache. Para isso, você precisa implementar alguma estrutura interna de dados no proxy para registrar quais objeto estão no cache e onde eles se encontram no disco. Você pode colocar essa estrutura de dados na memória principal; não é necessário fazê-la persistir após processos de desligamento.