Instituto de Matemática e Estatística da USP

MAC0352 - Redes de Computadores e Sistemas Distribuídos - 1s2022

EP2
Entrega até 8:00 de 31/5/2022
(INDIVIDUAL ou DUPLA)

Prof. Daniel Macêdo Batista

Servidor

Cliente

1 Problema

Neste EP você deverá implementar um sistema distribuído que possibilite uma partida de jogo da velha[1] em uma arquitetura híbrida (P2P e cliente/servidor) com tolerância a algumas falhas. O sistema deve ser composto por diversas máquinas em uma rede local (a correção será feita com 3 máquinas). A invocação do primeiro código (servidor) deve ser feita recebendo como parâmetro apenas a porta (ou portas, caso seja necessário definir mais de uma porta) na qual ele irá escutar por conexões dos clientes. A invocação do segundo código (cliente) deve ser feita passando como parâmetro o endereço IP e a porta (ou portas) do servidor. O servidor precisa suportar tanto conexões TCP quanto UDP. Os clientes precisarão informar o protocolo desejado como um parâmetro a mais, além do IP e da porta (ou portas).

O servidor será responsável por monitorar se os clientes estão conectados, autenticar os usuários, manter uma tabela de classificação dos jogadores (cada vitória dá 3 pontos. Cada empate dá 1 ponto) e registrar diversas ações em um arquivo de log. Note que os usuários do sistema e a tabela de classificação precisam ser persistentes. Ou seja, você vai precisar criar arquivos que mantenham essas informações para que elas sejam recuperadas na próxima vez que o servidor for executado. Os clientes executarão o jogo da velha propriamente dito, conectando-se entre si. Por isso que o jogo segue uma arquitetura híbrida. Os clientes conversam com o servidor para algumas ações (cliente/servidor) e depois conectam-se entre si para realizarem uma partida (P2P). Durante a partida, os comandos do jogo entre os clientes **devem** ser enviados exclusivamente entre os clientes. O servidor registrará algumas informações mas ele **não** pode ter a tarefa de receber os lances do jogo de um cliente e enviar para o outro. Sobre o protocolo da camada de transporte, a comunicação direta entre cliente e servidor pode ser tanto UDP ou TCP, no sentido de que o servidor deve ter condições de aceitar clientes em qualquer um desses dois protocolos (Não é para ser implementado o suporte a apenas um protocolo. **Tem** que ser implementado o suporte aos dois protocolos). Já a comunicação entre clientes, precisa ser TCP.

Então o servidor não pode ser um servidor http?

[1] https://pt.wikipedia.org/wiki/Jogo_da_velha

Esse EP é sobre redes e não sobre análise ou complexidade de algoritmos. Portanto, não se preocupe em implementar o melhor algoritmo para verificar o vencedor no jogo da velha. Embora não vá ser descontado ponto por algoritmos ruins, não abuse. Um código de dezenas de milhares de linhas com um monte de switch-case comparando todas as possíveis configurações do tabuleiro não é o que se espera de alguém que faz Ciência da Computação no IME-USP.

2 Requisitos

Dois códigos precisam ser implementados: um para o cliente e um para o servidor. Quando o cliente for executado ele deve exibir o seguinte prompt para o usuário:

```
JogoDaVelha>
```

O servidor deve executar sem necessidade de interação. O ideal é que ele seja um *daemon*[2], embora isso não seja obrigatório. É aceitável que ele funcione no segundo plano sendo invocado com '&' no shell.

O seu sistema deve implementar um protocolo de rede que atenda aos seguintes requisitos:

- verificação periódica, iniciada pelo servidor, de que os clientes continuam conectados. Esse mecanismo existe em diversos sistemas e é chamado de *heartbeat*;
- verificação periódica, entre clientes, da latência entre eles durante uma partida;
- envio das credenciais de usuário e senha em texto plano[3];
- troca de mensagens em modo texto entre cliente e servidor e entre clientes.

O protocolo criado por você deve usar comandos em ASCII para todas as ações, permitindo uma depuração fácil com o `wireshark` ou com o `tcpdump`. Comandos para as seguintes ações devem ser implementados:

1. *heartbeat* entre servidor e clientes
2. verificação de latência entre clientes numa partida (isso deve ser automático)
3. criação de um novo usuário
4. login
5. mudança de senha
6. logout
7. solicitação da lista dos usuários conectados
8. início de uma partida (nesse momento é necessário definir quem é 'X' e quem é 'O')

Isso deve ser feito entre clientes?
Isto é, sem a interação com o servidor?

[2] https://pt.wikipedia.org/wiki/Daemon_(computa%C3%A7%C3%A3o)

[3] No mundo real não se faz assim. O correto é utilizar criptografia (na maioria das vezes criptografia assimétrica). Quem quiser saber mais sobre isso, recomendo cursar a disciplina MAC0336

(isso deve ser automático)

9. envio de uma jogada (qual linha e qual coluna)
10. encerramento de uma partida antes dela terminar, por iniciativa de um dos jogadores
11. recebimento do tabuleiro atualizado (toda vez que alguém faz um lance, o tabuleiro precisa aparecer atualizado para ele e para o oponente. Nesse momento o shell do jogo deve travar para quem fez a jogada e ser 'liberado' apenas quando o outro jogador fizer o lance dele ou se a conexão com o oponente for interrompida por uma falha na rede)
12. envio do resultado da partida para o servidor (se o servidor 'caiu' no meio da partida, é necessário esperar para tentar reconectar e reenviar o resultado. Essa tentativa deve esperar até 3 minutos. Se nesse intervalo não for possível reconectar ao servidor, uma mensagem de erro deve ser informada para o jogador)
13. solicitação da classificação de todos os usuários existentes

Outros comandos podem ser implementados caso você ache necessário.

Os comandos 1, 2, 11 e 12 devem ocorrer sem necessidade de interação dos usuários. Eles serão enviados entre as entidades do sistema de forma periódica (1 e 2) ou quando ocorrer algum evento que faça eles serem necessários (11 e 12).

Os demais comandos precisam ser invocados pelos usuários nos prompts do jogo das seguintes formas:

- `new <usuario> <senha>`: cria um novo usuário
- `pass <senha antiga> <senha nova>`: muda a senha do usuário
- `in <usuario> <senha>`: usuário entra no servidor
- `halloffame`: informa a tabela de pontuação de todos os usuários registrados no sistema
- `l`: lista todos os usuários conectados no momento e se estão ocupados em uma partida ou não
- `call <oponente>`: convida um oponente para jogar. Ele pode aceitar ou não
- `play <linha> <coluna>`: envia a jogada
- `delay`: durante uma partida, informa os 3 últimos valores de latência que foram medidos para o cliente do oponente
- `over`: encerra uma partida antes da hora
- `out`: desloga

(ninguém ganha o jogo)

- `bye`: finaliza a execução do cliente e retorna para o shell do sistema operacional

A depender do resultado de um comando, alguns comandos não poderão ser usados em sequência. Por exemplo, se o usuário errar a senha, não faz sentido ele conseguir rodar o `out` ou o `in` já que ele não logou. Máquinas de estado[4] podem ser usadas para limitar os comandos que um usuário pode executar a depender do resultado do comando anterior.

[4] https://pt.wikipedia.org/wiki/M%C3%A1quina_de_estados_finita `https://uspdigital.usp.br/jupiterweb/obterDisciplina?sgldis=MAC0414`

O servidor precisa manter um arquivo de log informando tudo que aconteceu durante o tempo em que o código ficou rodando. Esse arquivo de log deve informar o momento do evento e qual foi o evento. Alguns eventos que não podem deixar de serem registrados são:

- Servidor iniciado (Informando se a última execução dele foi finalizada com sucesso ou se houve uma falha. Caso houve falha, e se havia alguma partida em execução, ele deve retomar o "controle" dessa partida passando a enviar os *heartbeats* para os clientes, caso eles ainda estejam conectados entre eles)
- Conexão realizada por um cliente (Endereço IP do cliente);
- Login com sucesso ou não (Nome do usuário que conseguiu, ou não, logar, e endereço IP de onde veio o login);
- Desconexão realizada por um cliente (Endereço IP do cliente);
- Início de uma partida (Endereço IP e nomes dos usuários dos jogadores);
- Finalização de uma partida (Endereço IP, nomes dos usuários dos jogadores e nome do vencedor);
- Desconexão inesperada de um cliente, verificada pelos *heartbeats* (Endereço IP do cliente);
- Servidor finalizado

O sistema deve tolerar a seguinte falha do servidor, limitada a um intervalo de 3 minutos. Se em 3 minutos o servidor não voltar a um estado correto de execução, a falha deve ser informada para algum cliente que esteja aguardando a falha ser corrigida para se comunicar com o servidor:

- Processo do servidor foi finalizado por um 'kill -9'

Recomenda-se a leitura das seções dedicadas a falhas no livro do Stevens[5]. Essa leitura pode ajudar na implementação do tratamento a essa falha.

2.1 Linguagem

Os programas podem ser escritos em qualquer linguagem de programação, desde que exista compilador/interpretador gratuito para GNU/Linux, e devem funcionar no shell, sem interface gráfica. Certifique-se de que seu programa funciona no GNU/Linux pois ele será compilado e avaliado apenas neste sistema operacional.

Você não pode utilizar bibliotecas, classes ou similares que já implementem o jogo da velha ou um sistema distribuído similar ao pedido. Códigos que não respeitem esse requisito terão nota ZERO.

[5] Seções finais do Capítulo 5 do livro Unix Network Programming, Volume 1: The Sockets Networking API (3rd Edition), Addison-Wesley 2011

2.2 Slides

Você deverá entregar, além dos códigos, um .pdf com slides que você usaria caso você fosse apresentar o seu trabalho. Os slides deverão apresentar o seu protocolo, informando se ele foi baseado em algum protocolo existente ou se você criou ele do zero. Outras decisões de projeto que você julgar que merecem ser apresentadas também podem ser incluídas. Além das informações sobre o protocolo, os slides também devem apresentar gráficos de análise de desempenho comparando o desempenho do sistema distribuído, em termos de uso de rede e de CPU, tanto do servidor quanto dos clientes, em três cenários: (i) apenas com o servidor, sem nenhum cliente conectado; (ii) com o servidor e com dois clientes conectados mas sem jogar e (iii) com o servidor e dois clientes conectados e jogando. Nos slides inclua informações sobre o ambiente computacional e de rede que você usou para realizar os experimentos e como você avaliou a carga na rede (Certifique-se de manter todos os outros softwares que utilizem a Internet fechados enquanto estiver realizando os experimentos). Os experimentos devem fazer a comunicação entre três computadores diferentes, ou seja, não podem ser feitos utilizando o localhost. Note que você não precisa ter 3 computadores reais para ir testando o EP e para fazer os experimentos. Recomenda-se fortemente o uso de máquinas virtuais via `VirtualBox`, `Xen` ou `docker`. Todos os resultados precisam ter validade estatística. Logo, faça uma quantidade razoável de medições e apresente os resultados com média e alguma informação de dispersão, como desvio padrão ou intervalo de confiança por exemplo.

Esses slides não serão apresentados mas considere que eles seriam apresentados em um tempo máximo de 10 minutos. Portanto, antes de enviar seu .pdf, faça um ensaio da apresentação para garantir que a quantidade de conteúdo cabe dentro de 10 minutos de apresentação.

Entregas sem o .pdf da apresentação não serão corrigidas e receberão nota ZERO.

3 Entrega

Você deverá entregar um arquivo .tar.gz contendo os seguintes itens:

- fonte do cliente e do servidor;
- Makefile (ou similar);
- arquivo LEIAME;
- .pdf dos slides.

O desempacotamento do arquivo .tar.gz deve produzir um diretório contendo os itens. O nome do diretório deve ser ep2-membros_da_equipe. Por exemplo: ep2-joao-maria no caso de dupla ou ep2-joao_dos_santos no caso de um EP individual.

A entrega do .tar.gz deve ser feita no e-Disciplinas.

O EP pode ser feito individualmente ou em dupla.

Obs.1: Serão descontados 2,0 pontos de EPs com arquivos que não estejam nomeados como solicitado ou que não criem o diretório com o nome correto após serem descompactados. Confirme que o seu .tar.gz está correto, descompactando ele no shell (não confie em interfaces gráficas na hora de testar seu .tar.gz pois alguns gerenciadores de arquivos criam o diretório automaticamente mesmo quando esse diretório não existe. Se você nunca usou o comando tar, leia a manpage e "brinque" um pouco com ele para entender o funcionamento).

Obs.2: A depender da qualidade do conteúdo que for entregue, o EP pode ser considerado como não entregue, implicando em MF=0,0. Isso acontecerá por exemplo se for enviado um .tar.gz corrompido, ou códigos fonte vazios.

Obs.3: O prazo de entrega expira às 8:00:00 do dia 31/5/2022.

4 Avaliação

60% da nota será dada pela implementação, 10% pelo LEIAME e 30% pelos slides. Os critérios detalhados da correção serão disponibilizados apenas quando as notas forem liberadas.