Instituto de Matemática e Estatística da USP

MAC0352 - Redes de Computadores e Sistemas Distribuídos - 2s2023

EP2
Entrega até 8:00 de 6/11/2023
(INDIVIDUAL)

Prof. Daniel Macêdo Batista

1 Problema

Neste EP você deverá implementar um sistema distribuído que possibilite uma partida multi-player do Pac-Man [1] em modo texto 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 (a pontuação é representada pela quantidade de pac-dots comidos) 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 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 quanto TCP, no sentido de que o servidor deve ter condições de aceitar clientes em qualquer um desses dois protocolos

[1] `https://youtu.be/i_OjztdQ8iw`

(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.

2 Requisitos

A arena a ser considerada no jogo é esta abaixo:

```
******.**... .....**.******
******.**.*******.**.******
******.**.*.. ..*.**.******
.....  ....*.....*.......F..
******.**.*.. ..*.**.******
```

onde:

- `*` representa uma parede;
- `.` representa um pac-dot;
- `F` representa um fantasma local, com a lógica de movimentação definida pelo jogo;
- Espaço em branco representa uma posição sem qualquer dos elementos acima.

Quando possível, por conta das paredes, a arena é cíclica (Na situação inicial da arena, se o fantasma local iniciar indo para a direita por três vezes consecutivas, ele surge no canto esquerdo da arena). Ao iniciar uma partida, o Pac-Man deve ser representado pelo caracter `C` e sempre deve iniciar no centro da arena. Quando um oponente ingressar em uma partida em andamento, ele deve ser representado com o caracter `f` e pode aparecer em qualquer posição sem elementos. Em momentos em que os dois tipos de fantasmas (`F` e `f`) estejam na mesma posição, o caracter `H` deve ser usado. A ordem de ações sempre é:

1. fantasma local primeiro
2. verifica se houve colisão com Pac-Man
3. fantasma remoto em seguida
4. verifica se houve colisão com Pac-Man
5. Pac-Man se movimenta
6. verifica se comeu algum pac-dot e se houve colisão com algum fantasma

Entre as ações 6 e 1, ao entrar no loop, implemente uma espera de 1 segundo para atualizar a tela a fim de permitir que os jogadores entendam o que aconteceu na ação anterior. Para ter uma noção do tipo de interação esperada para o jogo, assista ao vídeo em `https://youtu.be/pfBQZ3XSomI`, em que uma partida em andamento é exibida (o prompt não está sendo exibido nessa simulação. Antes do vídeo começar, a última movimentação realizada foi do fantasma local).

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:

```
Pac-Man>
```

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`. Comandos para as seguintes ações devem ser implementados:

1. *heartbeat* entre servidor e clientes
2. verificação de latência entre clientes numa partida
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 (quem inicia uma partida sempre é o Pac-Man)
9. participação de uma partida em andamento (quem entra na partida em andamento sempre será um fantasma)
10. envio de uma jogada (para qual direção movimentar o personagem)
11. encerramento de uma partida antes dela terminar, por iniciativa de um dos jogadores
12. recebimento da arena atualizada (toda vez que alguém faz um movimento, a arena precisa aparecer atualizada para ele e para o oponente. Nesse momento o shell do jogo deve travar para quem fez a jogada e ser 'liberado' apenas quando for a vez desse jogador informar seu movimento ou se a conexão com o oponente for interrompida por uma falha na rede)

[2] `https://tinyurl.com/mv7e63yy`

[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

13. 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 20 segundos. Se nesse intervalo não for possível reconectar ao servidor, uma mensagem de erro deve ser informada para o jogador)

14. solicitação da classificação de todos os usuários existentes

Outros comandos podem ser implementados caso você ache necessário.

Os comandos 1, 2, 12 e 13 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 (12 e 13).

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

- `novo <usuario> <senha>`: cria um novo usuário
- `senha <senha antiga> <senha nova>`: muda a senha do usuário
- `entra <usuario> <senha>`: usuário loga no servidor
- `lideres`: informa a tabela de pontuação de todos os usuários registrados no sistema
- `l`: lista todos os usuários conectados no momento e se estão jogando ou não
- `inicia`: o jogador inicia uma nova partida como Pac-Man
- `desafio <oponente>`: entra na partida sendo jogada pelo outro jogador. Nesse caso, entrará como fantasma (é permitido no máximo 1 fantasma remoto por partida, além do fantasma local)
- `move <direcao>`: movimenta o Pac-Man, ou o fantasma, na direção informada (as opções são `a` - esquerda, `s` - baixo, `d` - direita e `w` - cima)
- `atraso`: durante uma partida com outro oponente, informa os 3 últimos valores de latência que foram medidos para esse outro oponente. Se não tiver oponente, não precisa retornar nada.
- `encerra`: encerra uma partida antes da hora
- `sai`: desloga
- `tchau`: 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 `sai` ou o `desafio` já que ele não foi autenticado. Máquinas de estado[4][5] podem ser usadas para limitar os comandos que um usuário pode executar a depender do resultado do comando anterior.

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:

[4] `https://tinyurl.com/mv45tees`

[5] `https://uspdigital.usp.br/jupiterweb/obterDisciplina?sgldis=MAC0414`

- 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 nome do usuário);
- Entrada e saída de fantasma da partida existente (Endereço IP e nome do usuário)
- Finalização de uma partida (Endereços IP, nomes dos usuários 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 20 segundos. Se em 20 segundos o servidor não voltar a um estado correto de execução, todas as partidas em andamento devem ser finalizadas e usuários deslogados:

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

De forma similar, se algum cliente for morto com 'kill -9', o servidor deve considerar que ele foi desconectado e a partida que estava em andamento, com esse cliente, deve ser finalizada caso ele fosse um Pac-Man ou pode continuar sem a presença dele, caso ele fosse um fantasma.

Recomenda-se a leitura das seções dedicadas a falhas no livro do Stevens[6]. 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 um jogo de Pac-Man ou um sistema distribuído similar ao pedido. Códigos que não respeitem esse requisito terão nota ZERO.

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

2.2 Vídeo

No LEIAME do seu EP você deverá informar o link para um vídeo de no máximo 5 minutos em que você mostra três terminais com o seu jogo funcionando. Mostre a compilação dos códigos, caso você use uma linguagem compilada, inicie o servidor, inicie dois clientes, inicie uma partida com um cliente, entre na partida com outro cliente, finalize a partida e encerre a execução de todos os códigos corretamente. Durante a partida, mantenha na janela o painel do `wireshark` com a captura de todos os pacotes do seu sistema distribuído. Essa demonstração em vídeo pode ser feita no localhost. O vídeo pode ser colocado em qualquer plataforma gratuita como YouTube ou Vimeo por exemplo. Use legendas no vídeo explicando o que está acontecendo, com ênfase aos pacotes sendo exibidos no `wireshark`. Você também pode narrar ao invés de usar legendas, caso prefira.

3 Entrega

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

- fonte do cliente e do servidor;
- Makefile (ou similar);
- arquivo LEIAME (além de todas as informações necessárias para um bom LEIAME, não esqueça de colocar o link para o vídeo);

O desempacotamento do arquivo .tar.gz deve produzir um diretório contendo os itens. O nome do diretório deve ser ep2-seu_nome. Por exemplo: ep2-joao_dos_santos.

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

O EP deve ser feito individualmente.

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 6/11/2023.

4 Avaliação

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

Obs.: a correção não será feita com conexões no localhost (127.0.0.1) mas sim com os processos rodando em máquinas distintas em uma rede local. Certifique-se de que seu programa funciona corretamente mesmo em cenários onde as conexões não venham do localhost. Lembre-se que você não precisa ter três computadores para isso. Você pode usar máquinas virtuais.