Apresentação - EP2 MAC0352

Prof. Daniel Macedo Batista

Nome: Lucas Hiroshi Hanke Harada NUSP: 10737071
Nome: Lucas Paiolla Forastiere NUSP: 11221911

Protocolo

Estrutura geral dos pacotes

- Nossos pacotes são bastante simples, possuindo um cabeçalho fixo com apenas dois campos e um payload contendo informações em uma determinada ordem fixa;
- O primeiro byte do cabeçalho indica o tipo do pacote e é utilizado para saber qual é o tipo que estamos recebendo ou enviando;
- O segundo e terceiro bytes do cabeçalho indicam o tamanho restante e são utilizados para saber quantos bytes o payload possui;
- No total, nós criamos vinte tipos de pacotes, mas todos são bastante simples.
- Os únicos tipos de dados que passamos nos pacotes são inteiros ou strings. Para os inteiros, existem dois tipos: os de dois bytes e os de um byte apenas. As strings são escritas todas na mesma convenção: primeiro deixamos dois bytes para indicar o tamanho da string e os bytes seguintes são o valor dela.

Decisões de projeto

Estrutura do projeto

- Os principais arquivos do cliente e servidor são respectivamente ep2-client.cpp e ep2-server.cpp;
- Os pacotes foram implementados nos arquivos packages.cpp e packages.hpp;
- As principais funcionalidades do cliente e servidor foram implementadas respectivamente em client-functionality e server-functionality. Entretanto, as funcionalidades referentes ao jogo da velha em si foram implementadas nos arquivos ttt-engine.cpp e ttt-engine.hpp;
- A escrita e leitura de arquivos pelo servidor foram implementadas nos arquivos server-io.cpp e server-io.hpp;
- A criptografia de mensagens foi implementada em client-cript.hpp, client-cript.cpp, server-cript.hpp e server-cript.cpp;
- Utilidades gerais utilizadas no código foram implementadas em util.cpp e util.hpp;
- O log é salvo no arquivo ttt.log e o banco de dados de usuários é salvo em users.db.

Estrutura do servidor

- Assim como visto em aula, o servidor fica escutando por meio de uma porta conexões de clientes. Assim que uma nova conexão é feita, o servidor cria um processo filho para cuidar desse novo cliente.
- Cada processo filho possui três threads que executam ações diferentes: “heartbeat_handler_thread”, “invitation_handler_thread” e “entrada_handler_thread”.
    - A primeira é a responsável por enviar os heartbeats para o cliente a cada 10 segundos. Após o envio, ele espera os 10 segundos e se passados esse tempo não houve resposta, ele avisa que o cliente morreu e mata o processo filho que cuida desse cliente.
    - A segunda é o responsável por verificar se houve algum convite para o cliente que ele está cuidando. Existe uma variável chamada de “client_invitation” para cada cliente que vai indicar se houve ou não convite e essa thread verifica se houve mudança nessa variável(mais detalhes mais adiante)
    - A terceira é a responsável por receber pacotes do cliente, processá-los e tomar as ações necessárias referente a cada um dos diferentes tipos de pacote

Log e banco de dados

- Escrevemos no log em diversas ocasiões além das pedidas pelo professor. Por exemplo, quando o usuário se desloga ou quando há um troca de senha;
- Para escrever algo no log, basta passar um tipo de operação log_t e uma estrutura log_struct_t preenchida com os campos corretamente de acordo com a operação de log utilizada (por exemplo, quando um cliente conecta, precisamos indicar qual é o seu IP, para poder salvar no log);
- Para salvar os usuários, se utilizou uma classe user_t que armazena o nome, a senha, a pontuação, se ele está conectado, se ele está em partida, o ip e port e um inteiro especial para tratar do convite de usuários para jogar;
- No banco de dados, salvamos apenas o nome, senha, pontuação e as flags se ele está conectado e jogando. Fazemos isso pois o servidor atualiza o banco de dados sempre que uma operação de log é realizada, então se o servidor cair, temos um meio de recuperar quem eram os usuários conectados e o estado deles.

Estrutura do cliente

- O cliente possui duas threads: “ui_thread” e “entrada_thread”:
  - A primeira é a responsável por esperar comandos do usuário, processá-los e enviar pacotes para o servidor requisitando alguma resposta.
  - A segunda fica esperando pacotes do servidor e possivelmente responder para ele (caso dos heartbets) ou mostrar a resposta do servidor para o usuário.
- Quando o servidor cai, a cada 1 segundo o cliente tenta se reconectar. Ele faz isso durante 3 minutos e se não foi possível, ele somente encerra o processo. Se foi possível, ele envia um pacote para o servidor para mostrar quem ele é e volta tudo ao normal.

Conexão entre clientes

- Sejam C1 e C2 dois clientes e P1 e P2 os processos filhos do servidor que cuidam desses clientes respectivamente.
- Digamos que C1 queira convidar C2 para jogar. Então, C1 envia um pacote para P1 com o nome de C2 e depois, P1 avisa P2 que C1 quer jogar com C2. Esse aviso acontece da seguinte maneira:
    - Para cada cliente, existe uma variável chamada de “client_invitation” a qual todos os processos filhos têm acesso e que indicam se houve algum convite para este cliente e quem fez esse convite. Com isso, P1 atualiza essa variável do C2 indicando que C1 foi quem fez o convite. Ademais, P2 possui uma thread que a cada 1 segundo verifica se houve algum convite para C2 e assim que aparecer esse convite ele manda para C2.
    - O client_invitation também armazena a resposta do convite. Então, C2 envia para P2 a resposta e este atualiza o client_invitation avisando da resposta. Depois de enviar o convite, P1 fica verificando se houve resposta a cada 1 segundo e assim que a resposta aparecer ele envia para C1.

Conexão entre clientes

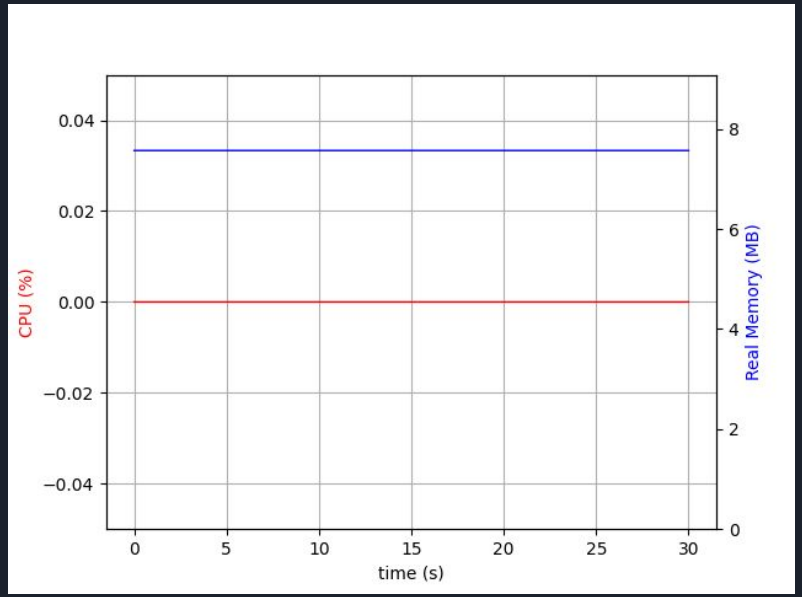
- Se o C2 aceitar o convite, ele procura uma porta vazia por meio da função “get_free_port” e enviará essa porta junto da resposta e ficará escutando por essa porta até o C1 requisitar uma conexão.
- Assim que C1 receber a resposta positiva junto da porta, ele pede faz a conexão com C2 por meio dessa porta e o jogo começa.
- Nesse jogo, assim como visto antes, existe uma thread chamada de “entrada_match_thread” que somente fica recebendo e processando esses pacotes, outra que chamei de “ui_match_thread” que é a responsável por interagir com o usuário e fica esperando inputs e enviando os movimentos para o outro jogador.
- Além disso, existe uma thread que chamamos de “latency_match_thread” que é a responsável por checar o delay. Ela envia um pacote de ping para o outro jogador e a thread da entrada aguarda a resposta e calcula o tempo de delay.
- Assim que o jogo acaba, cada cliente é responsável por enviar sua pontuação para o servidor

Análise de desempenho

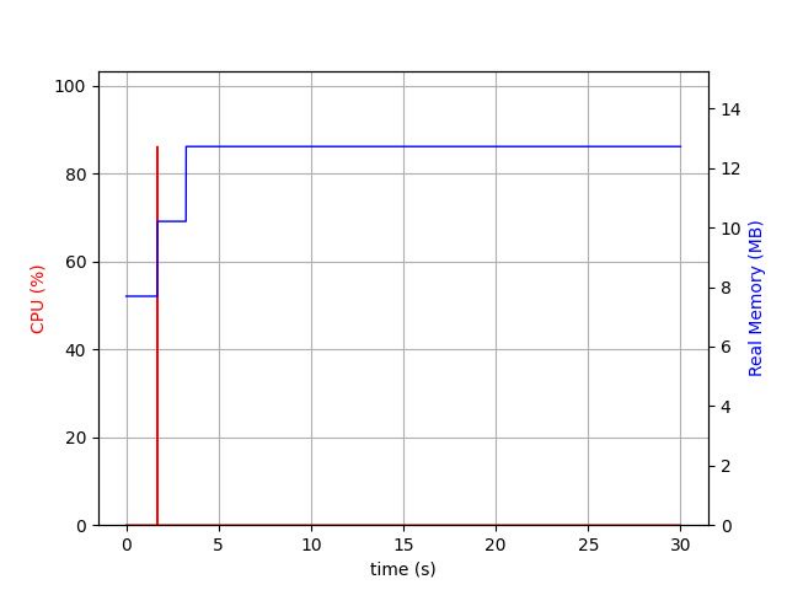
Configurações dos testes

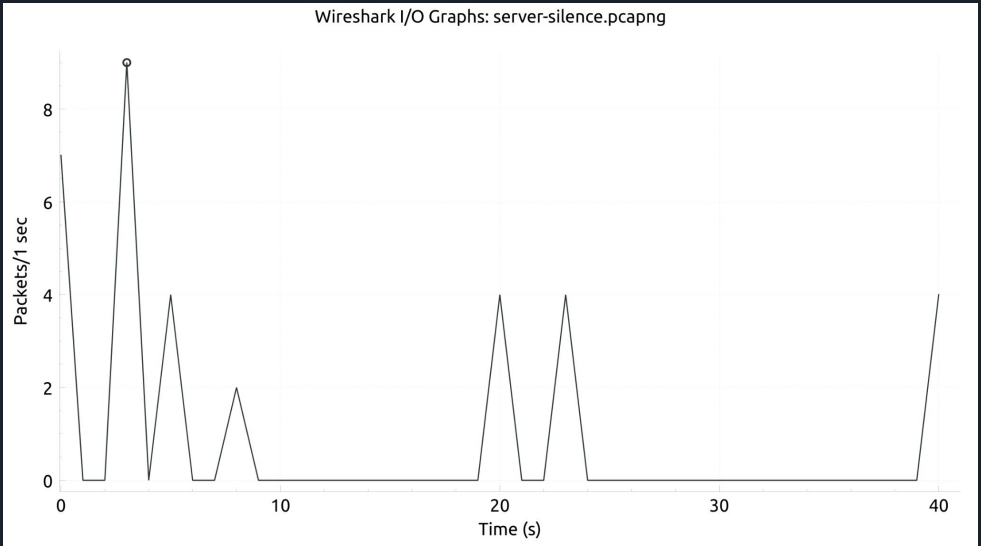
- Os testes foram realizados utilizando três máquinas virtuais utilizando o software VirtualBox 6.1.22. Uma delas, rodando o servidor, possui um sistema operacional Ubuntu 21.04, enquanto as outras duas possuem um sistema operacional Arch Kernel 5.12.11-arch1-1 (foram os clientes);
- A internet local é de 100 Mbps download e 60 Mbps upload, mas as três máquinas virtuais estavam abertas no mesmo computador host;
- **Observação:** houve um bug na hora de pegar o IP que às vezes ele acaba armazenando 0.0.0.0 ao invés do IP correto e não soubemos consertar ele. Na maioria das vezes acontece com o primeiro cliente, mas não há um critério claro. Por isso, pode haver um connect error na hora dos clientes começarem o jogo

Apenas com servidor sem nenhum cliente (servidor)

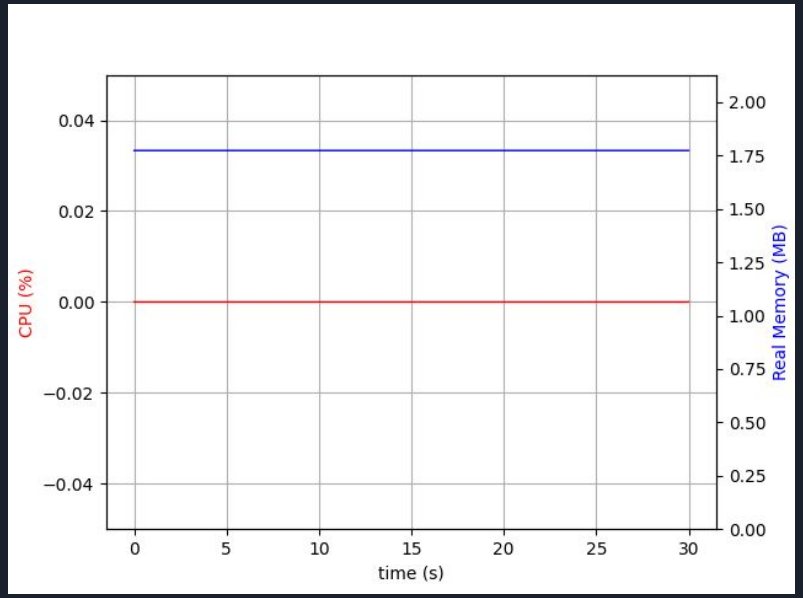


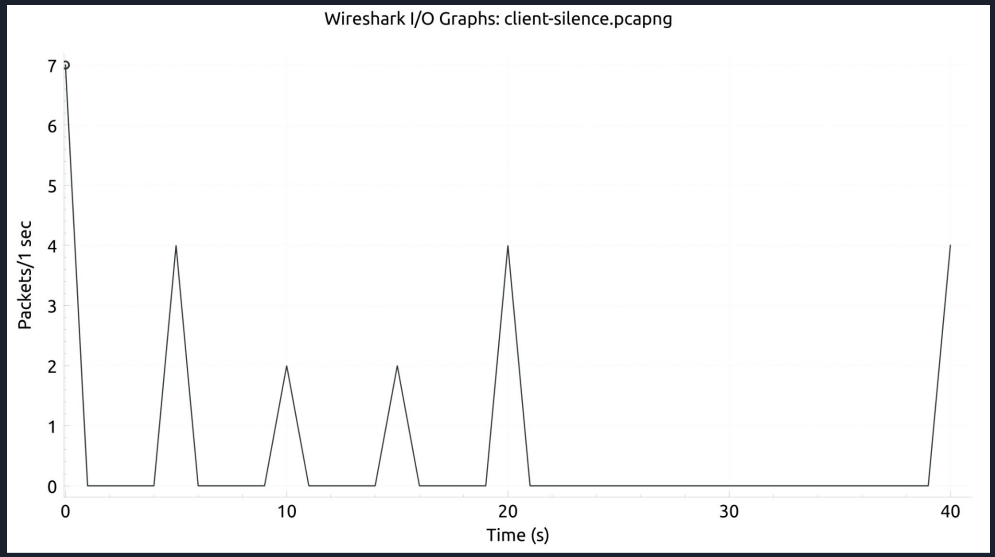
Com dois clientes conectados mas sem jogar (servidor)



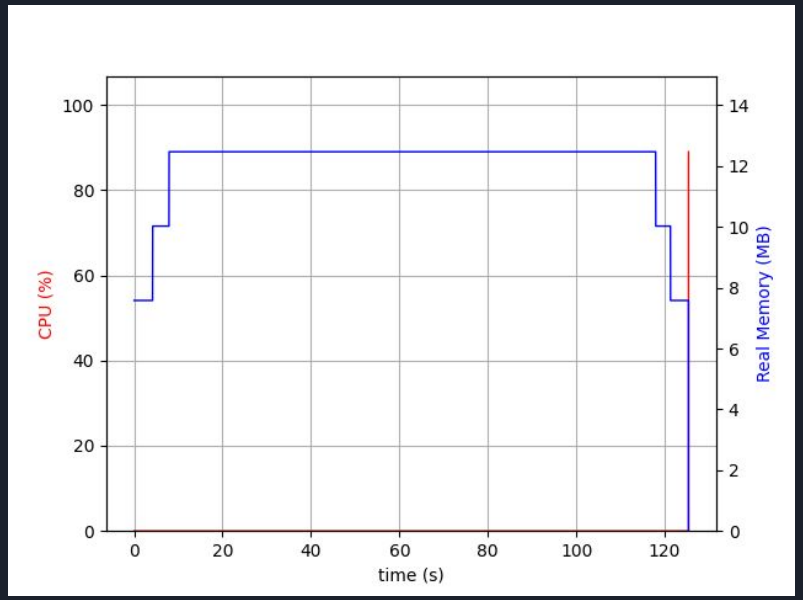


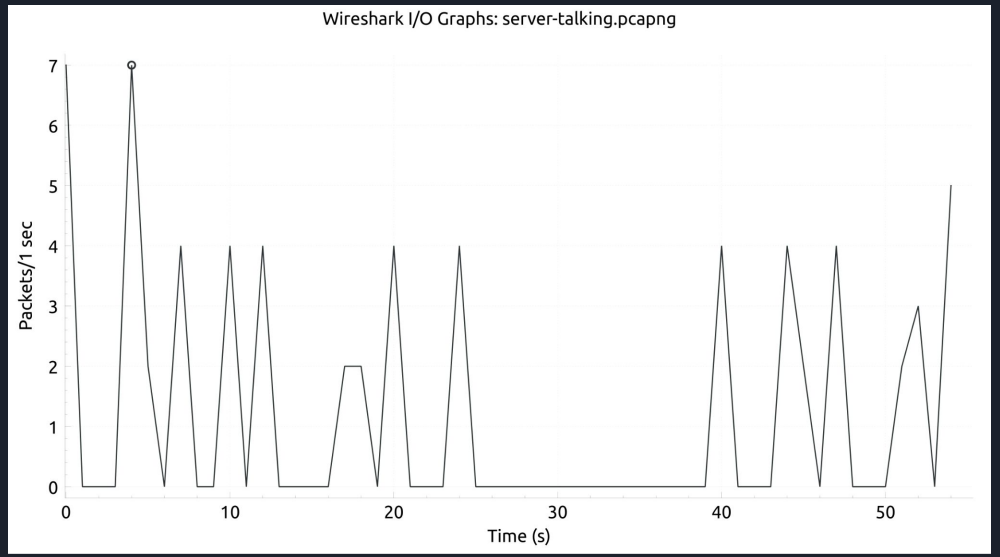
Com dois clientes conectados mas sem jogar (cliente)



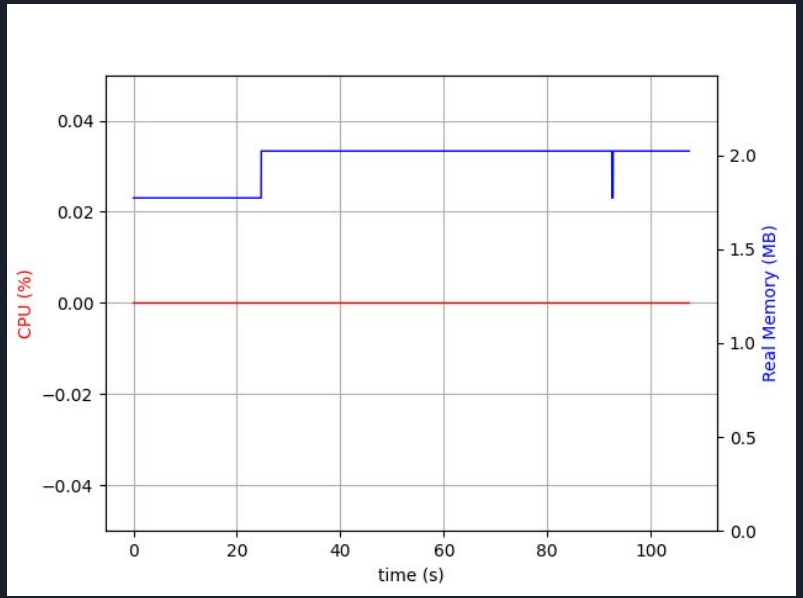


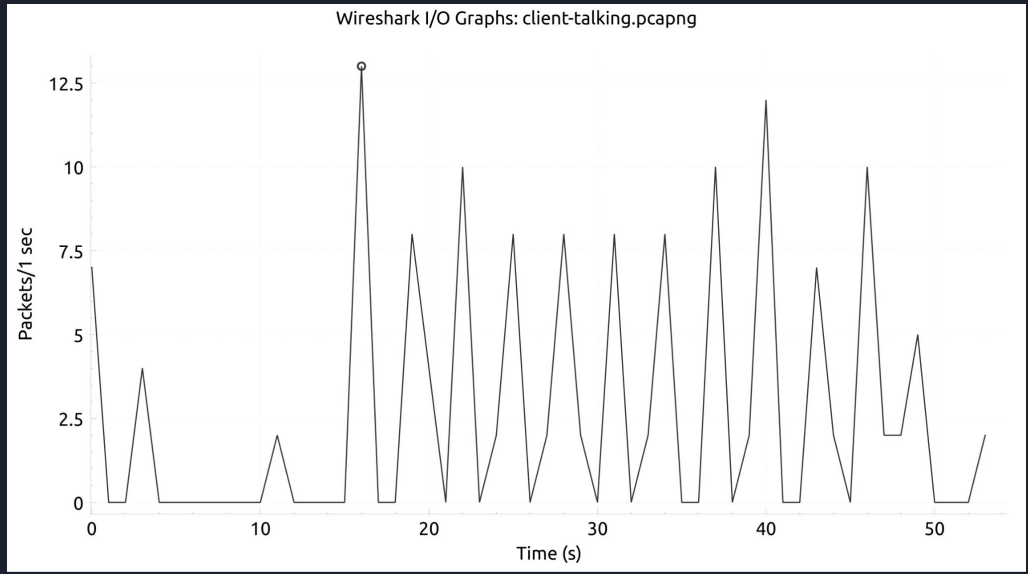
Com dois clientes jogando (servidor)





Com dois clientes jogando (cliente)





Comentários - Uso de CPU

- Note que boa parte do tempo o uso de CPU no servidor é sempre muito próximo de zero, pois o servidor não fica o tempo inteiro rodando. Existem algumas checagens periódicas como o heartbeat e a do “client_invitation”, mas o tempo entre uma e outra é razoavelmente grande. Nos gráficos do cliente ocorre a mesma coisa.
- Somente em alguns poucos momentos que há um pico no uso de CPU no servidor que é justamente quando os clientes se conectam e desconectam e o servidor deve trabalhar mais para destruir/construir os processos filhos, registrar no log e etc.

Comentários - Uso de Rede

- Quando o servidor não possui nenhum cliente, não existe absolutamente nenhum uso de rede. Por isso não exibimos nenhum gráfico;
- Quando o servidor está com dois clientes conectados, mas sem nenhuma operação por parte eles, o uso de rede é bem baixo, vemos que o servidor possui um pico maior quando os dois clientes se conectam e depois existem alguns picos mais baixos quando pingamos os clientes. Por parte do cliente, vemos o mesmo padrão: um pico maior no começo ao conectar e picos menores depois referentes aos pings;
- Quando os clientes jogam, entretanto, vemos um uso de rede mais intenso (mas ainda bem baixo). Por parte do servidor, temos mais ou menos o mesmo padrão de anteriormente. Já por parte do cliente, temos que depois de um tempo, observamos um uso grande da rede, que é quando os jogadores estão jogando e trocando mensagens entre si.