Implementação e Análise de Modelos de Servidores Web

Breno Esteves, Gabriel de Paula, Guilherme Francis

Universidade Federal de São João Del Rei
São João del Rei / MG - Brasil

{brenoesteves243,gabriel.meira.2004,guilherme2036}@aluno.ufsj.edu.br

Resumo. *Este trabalho apresenta o desenvolvimento de uma aplicação cliente-servidor para o jogo da forca, explorando diferentes modelos de servidores TCP. São implementados servidores iterativo, multi-thread, multi-thread com fila de tarefas, e concorrente, demonstrando as vantagens e desafios de cada abordagem. O sistema busca suportar múltiplos clientes simultâneos e utilizar estruturas de dados como filas e algoritmos para gerenciar o estado do jogo e a comunicação, destacando a relevância do protocolo TCP e do processamento paralelo para aplicações em redes de computadores.*

1. Introdução

Nos dias atuais, a comunicação em redes de computadores é essencial para o funcionamento de aplicações distribuídas, desde serviços bancários até plataformas de entretenimento. A capacidade de lidar com múltiplas conexões simultâneas e processar informações de maneira eficiente é um desafio frequente enfrentado por desenvolvedores. Este trabalho se insere nesse contexto ao propor uma aplicação cliente-servidor que, além de ser um exemplo didático, reflete a necessidade de criar sistemas que conciliem desempenho, escalabilidade e simplicidade no processamento de dados em rede.

O presente trabalho tem como objetivo o desenvolvimento de uma aplicação cliente-servidor para o jogo da forca utilizando o protocolo TCP (Transmission Control Protocol). O jogo permite que múltiplos clientes interajam com o servidor, que gerencia o estado do jogo e processa as entradas de cada jogador. O trabalho contempla a implementação de diferentes modelos de servidores, incluindo interativo, multi-thread e um servidor concorrente. Também explora conceitos fundamentais de redes de computadores, gerenciamento de threads e manipulação de sockets.

2. Desenvolvimento

A camada de transporte do TCP foi utilizada para o desenvolvimento desse trabalho, dado que esse protocolo é amplamente utilizado em aplicações que requerem comunicação confiável, sendo exatamente o que precisamos para que o jogo da forca funcione como o esperado. Ele garante a entrega ordenada e livre de erros dos dados enviados entre cliente e servidor.

Diferentes modelos de servidores foram implementados para ter um melhor entendimento da programação prática de protocolos da camada de aplicação através de um jogo da forca, popular por possibilitar que seja multiusuário colaborativo. Sendo assim, quatro tipos de servidor foram desenvolvidos usando técnicas de programação distintas, a fim de avaliá-las qualitativamente em diferentes critérios.

2.1. Servidor Iterativo

O servidor iterativo é a abordagem mais básica e direta para a comunicação cliente-servidor. Nesse modelo, apenas um cliente pode ser atendido por vez. Quando uma conexão é aceita, o servidor entra em um loop para processar as requisições desse cliente até que a comunicação seja encerrada. Apesar de simples, essa abordagem é limitada porque impede o atendimento de múltiplos clientes simultaneamente, o que pode resultar em gargalos em cenários de alta demanda. Este modelo é mais adequado para sistemas onde há um único cliente, como um dispositivo de automação doméstica comunicando-se com um servidor central.

Fazendo uma alusão ao código, após estabelecer a conexão, o servidor inicia a comunicação direta com o cliente, mantendo o vínculo até que o cliente se desconecte ou o jogo seja finalizado. Caso o cliente encerre a conexão, o socket é fechado, liberando os recursos associados. Isso permite que o servidor retorne ao estado de espera, pronto para aceitar novos clientes.

2.2. Servidor Multi-Thread

Neste modelo, cada cliente conectado ao servidor é gerenciado por uma thread exclusiva. Isso significa que o servidor pode atender vários clientes simultaneamente, dividindo os recursos computacionais disponíveis para as threads em atividade. Cada uma é responsável por processar a comunicação com um cliente específico, permitindo maior escalabilidade. Esse, por outro lado, é comumente utilizado em servidores de chat ou jogos multiplayer simples, onde cada cliente ou grupo de clientes tem sua própria sessão dedicada.

A implementação desse sistema para o jogo é relativamente simples, com uma thread sendo dedicada exclusivamente a cada cliente conectado, sendo destruída logo após o término de sua comunicação. Essa abordagem garante uma interação eficiente no começo, mas, à medida que o número de clientes aumenta, a criação de múltiplas threads pode levar a uma sobrecarga significativa no sistema, conforme será analisado na seção de testes.

2.3. Servidor Multi-thread com Fila de Tarefas

Esse modelo utiliza um pool de threads pré-criado para processar conexões de clientes. As conexões recebidas são armazenadas em uma fila e atendidas pelos threads disponíveis na pool. Essa abordagem evita a criação e destruição constante de threads, economizando recursos e melhorando o desempenho em sistemas com alta carga de conexões simultâneas. Amplamente utilizado em servidores web, onde um grande número de requisições precisa ser gerenciado de forma eficiente, como em plataformas de streaming.

A fila foi implementada como uma estrutura circular, otimizando a inserção e remoção de conexões de forma eficiente. Essa abordagem permite um gerenciamento contínuo e dinâmico dos recursos, sem desperdício de espaço. Além disso, a estrutura utiliza mutexes e variáveis de condição para garantir que os threads possam acessar a fila de maneira sincronizada, prevenindo condições de corrida e assegurando a integridade dos dados compartilhados.

2.4. Servidor Concorrente

A abordagem baseada em concorrência utiliza multiplexação de entrada e saída para monitorar múltiplos sockets simultaneamente. Esse modelo é eficiente e escalável em termos de recursos porque utiliza um único processo para gerenciar várias conexões. O Select verifica quais sockets têm dados prontos para leitura ou escrita e processa apenas o necessário. Muito usado em servidores de baixa latência, como proxies e balanceadores de carga, onde o foco é gerenciar até 1024 conexões simultâneas com eficiência.

No código, o servidor concorrente é configurado para monitorar o socket principal (de escuta) e os sockets de clientes conectados. A função FDSET é usada para adicionar os sockets ao conjunto monitorado, enquanto FDISSET verifica quais sockets estão prontos para leitura ou escrita. Quando um cliente se desconecta, o socket correspondente é removido do conjunto usando FDCLR e fechado para liberar os recursos. Isso garante que o servidor continue funcionando corretamente em caso de desconexões.

3. Descrição dos Algoritmos e Estruturas de Dados

Todos os servidores possuem uma finalidade em comum, respondendo às requisições da mesma forma, com a diferença de como lidam com os múltiplos clientes solicitando dados ao mesmo tempo.

3.1. Estruturas de Dados do Jogo

Os tipos abstratos de dados facilitam o gerenciamento do jogo, controlando seu estado e as palavras de maneira a não enviar a resposta correta ao cliente, evitando fraudes.

```
typedef int alphabet_state_t[26];
typedef struct {
    char chars[WORD_MAX_SIZE + 1];
} word_t;
typedef struct {
    int state;
    int lifes;
    alphabet_state_t alphabet;
} game_t;
```

3.2. Comunicação

Uma função específica para se comunicar com os clientes usando o protocolo HTTP é responsável por toda essa interação, lendo as mensagens enviadas pelo cliente através do socket associado à conexão. Criar estruturas de dados e funções separadas do código principal possibilita o reúso e, consequentemente, favorece a manutenibilidade dos sistemas.

Com base na mensagem recebida, a função processa as informações e atualiza o estado atual do jogo. Após processar os dados, a função envia uma resposta ao cliente, encapsulando o estado atualizado do jogo, incluindo informações sobre as letras corretas reveladas, as letras erradas tentadas e o progresso geral da palavra mistério.

```
int communicate_with_client(int client_sock, game_t* game, word_t*
mystery_word, word_t* correct_word);
```

3.3. Servidores

A seguir, são descritos os funcionamentos de cada modelo de servidor, detalhando os passos executados em cada caso, assim como as estruturas de dados especiais utilizadas.

3.3.1. Funcionamento do Servidor Iterativo

Algoritmo: O servidor iterativo segue um modelo sequencial e bloqueante. Ele aceita uma conexão de um cliente, processa completamente a solicitação recebida, envia a resposta e encerra a conexão antes de retornar ao estado de espera para aceitar um novo cliente. Este ciclo se repete indefinidamente enquanto o servidor está em execução.

Passos:

1. Criar e configurar o socket.
2. Colocar o socket em modo de escuta.
3. Aceitar uma conexão do cliente.
4. Processar a requisição recebida.
5. Enviar a resposta ao cliente.
6. Fechar o socket do cliente.
7. Retonar ao passo 2.

Estruturas de Dados: O servidor iterativo não utiliza estruturas de dados complexas, usando apenas o socket normalmente para se comunicar com o cliente atual.

3.3.2. Funcionamento do Servidor Multi-Thread

Algoritmo: Este modelo permite que cada conexão seja tratada de forma independente através da criação de processos ou threads. O processo principal aceita a conexão e delega o processamento a um subprocesso, enquanto retorna imediatamente para aceitar outras conexões.

Passos:

1. Criar e configurar o socket.
2. Colocar o socket em modo de escuta.
3. Aceitar uma conexão do cliente.
4. Criar um subprocesso para lidar com o cliente.
    (a) (Subprocesso) Processar a requisição recebida.
    (b) (Subprocesso) Enviar a resposta ao cliente.
    (c) (Subprocesso) Fechar o socket do cliente.
5. Retornar ao passo 2.

Estruturas de Dados: Há a criação de pthreads (threads POSIX) para lidar com os clientes de forma independente e concorrente.

3.3.3. Funcionamento do Servidor Multi-Thread com Fila de Tarefas

Algoritmo: O modelo de threads com fila de tarefas segue o padrão produtor-consumidor. O processo principal atua como o produtor, aceitando conexões e colocando-as em uma fila. Um conjunto fixo de threads, os consumidores, processa as conexões de forma concorrente, buscando um novo cliente para atender quando o trabalho acaba.

Passos:

1. Criar e configurar o socket.
2. Criar a fila de tarefas e inicializar os subprocessos do pool.
    (a) Os subprocessos estão prontos para retirar as conexões da fila em ordem.
3. Colocar o socket em modo de escuta.
4. Aceitar uma conexão do cliente e inserir na fila.
    (a) (Subprocesso) Processar a requisição recebida.
    (b) (Subprocesso) Enviar a resposta ao cliente.
    (c) (Subprocesso) Fechar o socket do cliente.
5. Retornar ao passo 3

Estruturas de Dados: Essa estrutura segue o padrão FIFO (First-In, First-Out), garantindo que as conexões sejam atendidas na ordem de chegada na fila. Mutex e Variáveis de Condição garantem a sincronização entre threads ao acessar a fila.

3.3.4. Funcionamento do Servidor Concorrente

Algoritmo: Este modelo utiliza a função select para monitorar múltiplos descritores de socket simultaneamente. Ele permite que o servidor identifique quais sockets estão prontos para leitura ou escrita e processe apenas esses, sem bloquear o restante.

Passos:

1. Criar e configurar o socket.
2. Adicionar o socket principal a um conjunto de descritores monitorados.
3. Usar o Select para identificar alguma novidade nos descritores.
    - Se um novo cliente conectar, adicionar o socket do cliente ao conjunto.
    - Se um cliente enviar dados, processar a requisição e enviar a resposta.
4. Remover sockets fechados do conjunto.
5. Retornar ao passo 3

Estruturas de Dados: Um conjunto de descritores (FDSET) permite ao servidor identificar quais sockets devem ser atendidos no momento.

4. Resultados e Análises

Um aspecto fundamental no estudo de sistemas de comunicação em rede é a avaliação de desempenho e eficiência. Neste estudo, propõe-se a utilização de métricas como o total de requisições com sucesso e falha e o tempo de resposta. Essa análise foi realizada utilizando a biblioteca Locust, uma ferramenta de código aberto para simulação de carga em sistemas distribuídos.

Todos os servidores foram testados ao extremo, variando o número de usuários que requisitam dados ao mesmo tempo em 1, 10 e 100. Assim, é possível testar os servidores em condições de baixa e alta concorrência, superando os recursos computacionais disponíveis em alguns casos. Para aumentar o custo de uma requisição também é enviado ao cliente um arquivo de 1 MB, apenas visando ter resultados mais aprofundados.

4.1. Total de requisições com Sucesso

A Figura 1 explora a relação entre o número de usuários simultâneos (clientes) conectados a diferentes servidores e o total de requisições com sucesso processadas por cada um deles em um intervalo de 30 segundos.

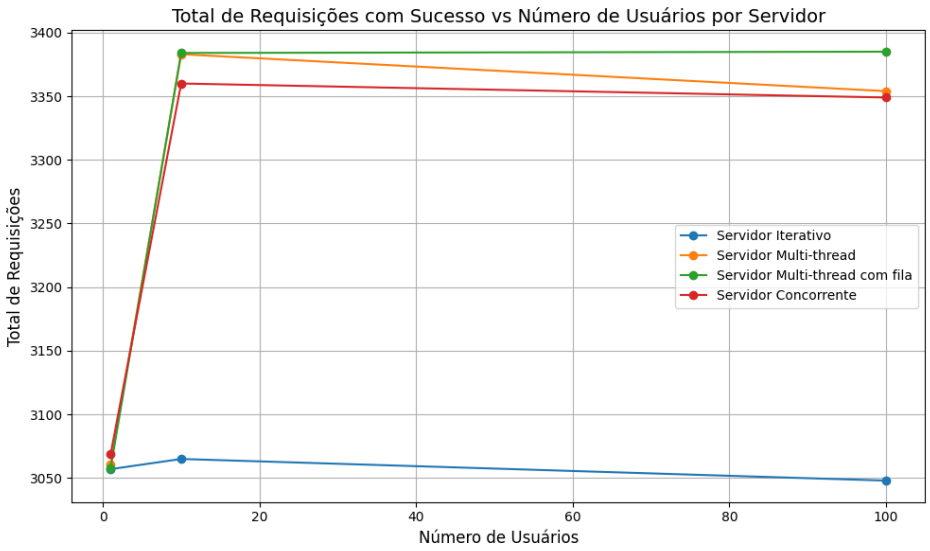

Figura 1. Total de requisições

É possível notar claramente que todos os servidores, para apenas um usuário, possuem um desempenho praticamente igual, com sutis diferenças devido às estruturas de dados mais e menos complexas utilizadas.

O Servidor Iterativo, representado pela linha azul, apresentou o pior desempenho. Esse comportamento era o esperado, visto que o modelo iterativo é bloqueante, processando apenas uma conexão por vez, o que limita severamente a escalabilidade e faz com que, independente da quantidade de usuários, o resultado fique estável.

Já o Servidor Multi-Thread, identificado pela linha laranja, mostra um desempenho inicial forte, entretanto, à medida que a carga aumenta, há uma leve queda no total de requisições processadas. Sugerindo que o modelo sofre devido ao aumento excessivo de threads, mostrando que o exagero nem sempre levará a bons resultados

O Servidor Multi-thread com Fila de Tarefas demonstrou ser a implementação mais eficiente para a quantidade total, processando consistentemente um alto número de requisições com sucesso, mesmo sob cargas elevadas. Similar ao anterior, o Servidor Concorrente também mostrou estabilidade, com um desempenho um pouco abaixo.

4.2. Total de requisições com Falha

Não basta apenas analisar as requisições bem sucedidas, é preciso investigar também as falhas para tentar de alguma se aproximar o máximo possível de zero. O gráfico da Figura 2 traz a quantidade total de falhas geradas no mesmo teste de 30 segundos.

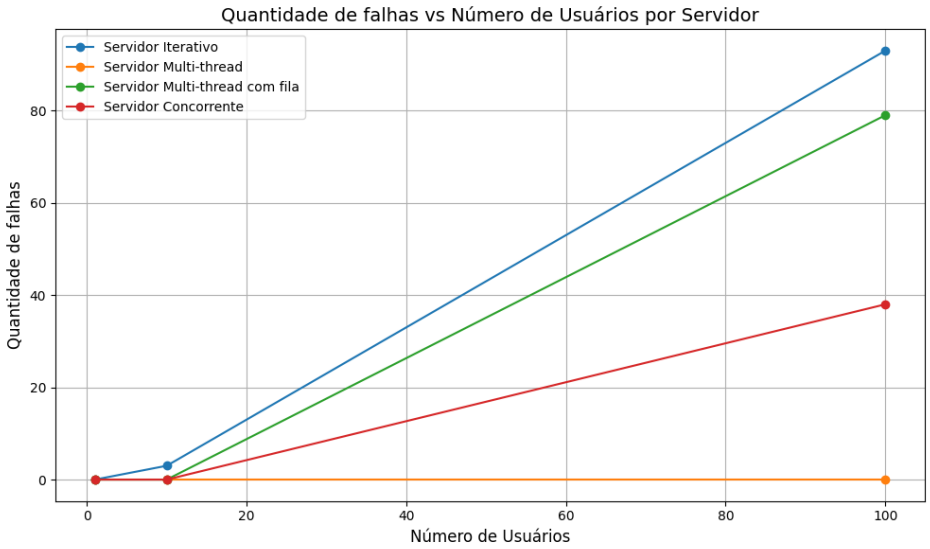

Figura 2. Total de requisições

O Servidor Iterativo, por sua definição, sempre terá dificuldades para lidar com requisições simultâneas, obtendo a pior marca no teste.

Por outro lado, o Servidor Multi-Thread demonstrou uma performance mais robusta, não apresentando falhas mesmo com um número elevado de clientes. No entanto, vale ressaltar que a criação de threads é limitada pelo sistema operacional. Uma vez atingido o limite de threads suportado, o servidor começa a falhar ao tentar criar novas threads, impactando diretamente a capacidade de atender novas requisições.

Outro fator determinante no desempenho do Servidor Multi-Thread é o tamanho da fila de tarefas. A quantidade de clientes na fila deve ser cuidadosamente planejada para garantir que o servidor consiga processar o volume de requisições dentro do tempo adequado, sem sobrecarregar o sistema. Observou-se um aumento claro no número de falhas conforme o tamanho da fila de tarefas se tornava inadequado.

O Servidor Concorrente também apresentou falhas em razão da falta de um gerenciamento paralelo dos descritores. Algumas requisições podem chegar enquanto outras estão sendo processadas, e o alto volume dificulta para o servidor receber corretamente todos. É preciso dar atenção a esse detalhe durante o desenvolvimento.

4.3. Tempo médio de resposta

A Figura 3 demonstra a relação entre o número de usuários conectados simultaneamente a diferentes servidores e o tempo médio de resposta para processar as requisições, sendo um ponto importante a ser analisado para aplicações que exigem velocidade na resposta.

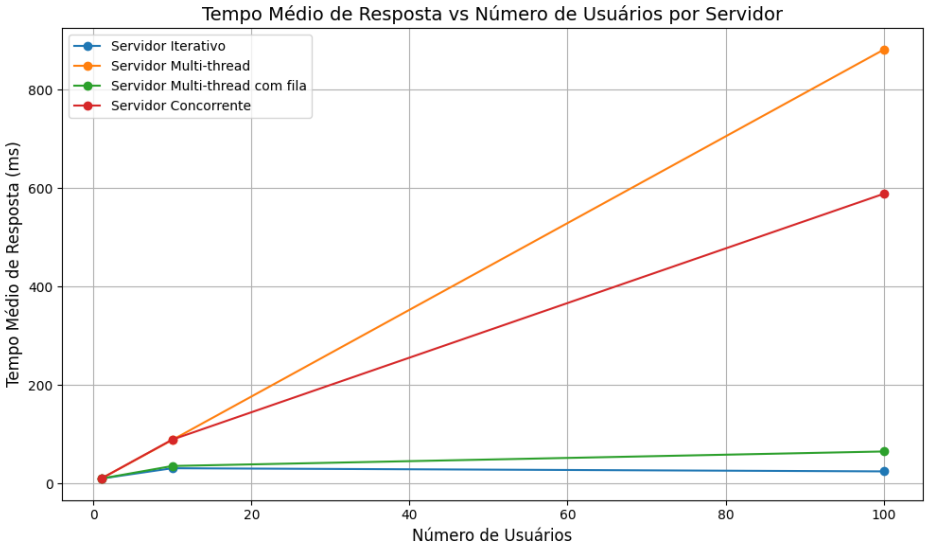

Figura 3. Gráfico do tempo médio de resposta

De forma geral, o gráfico revela padrões claros de desempenho entre os servidores. O Servidor Iterativo se destaca pelo menor tempo de resposta, isso se deve ao fato de que todos os recursos estão disponíveis exclusivamente para atender a um único cliente.

Em contrapartida, o Servidor Multi-Thread, por criar threads deliberadamente, faz com que os recursos sejam divididos entre todos os clientes que estão sendo atendidos, melhorando o tempo para atender a todos no geral, mas piorando consideravelmente o tempo individual.

Nota-se que o uso de uma quantidade limitada de subprocessos e uma fila de tarefas torna o tempo de resposta muito mais estável, representado pela linha verde, mesmo para muitos usuários.

Por fim, ao analisar o Servidor Concorrente, observa-se um aumento linear no tempo de execução, o que é compreensível, pois, assim como no modelo multi-thread, os recursos são compartilhados. No entanto, esse modelo não requer a criação e destruição frequente de subprocessos, o que melhora o desempenho.

5. Conclusão

Nesse projeto destaca-se o sucesso na implementação de uma aplicação cliente-servidor para o jogo da forca, explorando de forma abrangente diferentes modelos de servidores TCP. A utilização de servidores iterativo, multi-thread, multi-thread com fila de tarefas e concorrente permitiu a análise das características, vantagens e limitações de cada abordagem, evidenciando como soluções distintas podem atender a demandas específicas de aplicações em redes de computadores.

O servidor iterativo é claramente inadequado para lidar com usuários simultâneos, enquanto o servidor multi-thread puro sofre limitações em cenários de cargas muito altas devido ao aumento da quantidade de threads. A adição de uma fila de tarefas desse ser pensada levando em consideração o fluxo de clientes acessando a aplicação. Além disso, foi possível perceber que o servidor concorrente apresentou um bom desempenho, mas com uma leve degradação em situações de alta demanda. Esses resultados evidenciam como diferentes abordagens afetam a eficiência e a escalabilidade dos servidores.

O trabalho demonstrou que o protocolo TCP, com suas propriedades de entrega confiável e ordenada, é essencial para o funcionamento adequado de sistemas interativos e multiusuários. Ademais, a implementação de conceitos como multithreading, sincronização e manipulação de sockets contribuiu significativamente para a compreensão prática de tecnologias fundamentais para o desenvolvimento de aplicações distribuídas.

Ao final, pode-se concluir que o projeto atingiu os objetivos propostos, proporcionando um exemplo didático e funcional para a aplicação de princípios de redes de computadores. Este trabalho também ressalta a importância de soluções eficientes e escaláveis em sistemas baseados em redes, oferecendo uma base sólida para estudos futuros.

Referências

[1] Tanenbaum, Andrew S., and Wetherall, David J. (2010). *Redes de Computadores*. 5ª Edição. Pearson.

[2] Kurose, James F., and Ross, Keith W. (2010). *Redes de Computadores e a Internet: Uma Abordagem Top-Down*. 5ª Edição. Pearson.