Universidade Federal de São João Del Rei

Redes de Computadores
Trabalho Prático 2
Modelos de implementação de Servidores Web

Carlos Magno Geraldo Barbosa
Lucas Geraldo Silva Cruz

Prof. Dr. Rafael Sachetto Oliveira

São João Del Rei, Junho de 2017

1.Introdução **2**

2. Implementação **2**

2.1 Estrutura de Dados 2

2.2 Servidores 3

2.3 Servidor Iterativo 3

2.4 Servidor utilizando Fork 4

2.5 Servidor utilizando Threads e Fila 4

2.6 Servidor utilizando Select 4

3.Principais Rotinas 5

4.Testes e Análise de Resultados **5**

4.1 Resultados Obtidos 6

4.2 Análise de Resultados 7

5.Conclusão **8**

6.Referências Bibliográficas **8**

1.Introdução

Para prover uma estrutura de que facilita a implementação e atualização os protocolos de rede são divididos em uma estrutura de camadas. Esta arquitetura permite que sejam discutidas camadas específicas de um sistema grande e complexo, cada camada provê uma gama de serviços que utilizam os serviços da camada inferior e provê serviços para camada superior(Kurose e Ross).

Compondo uma parte da mais alta dessas camadas, a camada de aplicação, estão os servidores Web. Servidores web são softwares que executam em máquinas presentes na borda da rede mundial de computadores. Tais softwares fazem parte da camada de aplicação pois tem a função de responder requisições do protocolo http vindas de clientes espalhados por todo o mundo.

Neste trabalho serão especificados e implementados quatro modelos de implementação de servidores Web: iterativo, multi processos, com fila de tarefas e servidor concorrente, sendo que tais modelos se diferenciam nas técnicas utilizadas por cada um para responder às requisições recebidas. As requisições dos clientes serão simuladas pela ferramenta *siege,* por meio da qual é possível realizar testes para verificar o desempenho dos servidores.

2. Implementação

Todos os modelos de servidor implementados neste projeto foram desenvolvidos na linguagem C, utilizando a API Sockets da biblioteca da linguagem para possibilitar a comunicação dos mesmos com o cliente *Siege*. Para viabilizar transferência confiável de dados, os servidores adotaram os serviços providos pelo protocolo TCP da camada de transporte o qual implementa políticas que permitem o modo de transferência confiável desejado.

2.1 Estrutura de Dados

Para implementar os servidores web foram utilizadas estruturas de dados e tipos abstrato de dados, TADS, sendo as mesmas compostas pelos componentes *socket_clientes* e a TAD *endereco_servidor* e *endereco_cliente* instanciadas a partir da *struct sockaddr_in* já implementada na biblioteca *socket.* Nas TADS é onde são definidas as informações necessárias para conexão como: protocolo de transporte utilizado, número de porta do servidor e do cliente, e endereço das máquinas na rede.

2.2 Servidores

Em todos os quatro modelos de servidores implementados foram utilizadas as mesmas definições de propriedades básicas para que o servidor pudesse ser colocado no ar. Em todos eles foram feitas primeiramente a abertura do socket de comunicação, depois a atribuição deste socket a uma porta de comunicação, a definição do protocolo de transporte que o servidor iria utilizar, e o tamanho máximo da fila de entrada do socket (número máximo de conexões com clientes ativas simultaneamente).

Para facilitar os testes e a execução, todos os servidores foram unificados em um único código-fonte no qual é possível escolher qual modelo de servidor o usuário deseja executar. Além disso a cada requisição de um cliente é retornada uma mensagem padrão HTTP com formato destacado na figura 1.

Figura 1

```
HTTP/1.1 200 OK
Server: C3PO-web7
Connection: close
Content-length: 89
Content-Type: text/html
<!DOCTYPE html><html><head>
<title>Redes</title></head><body>
<h1>42...!!<h1></body></html>
```

A mensagem indica respectivamente: a versão do protocolo HTTP utilizada, um código de status, o tipo de documento enviado na resposta, e o conteúdo da resposta, no caso uma página HTML.

2.3 Servidor Iterativo

O servidor iterativo implementa um algoritmo que processa em uma única linha de execução, de forma que é atendida apenas uma requisição por vez respeitando a ordem em que cada cliente solicita uma conexão com o servidor.

Ao estabelecer uma conexão com um cliente, o servidor espera receber a requisição da página HTML. Assim que a recebe ele responde com a mensagem já descrita anteriormente, e após isso fecha a conexão com o cliente.

2.4 Servidor utilizando Fork

Neste modelo de implementação, a cada conexão de um cliente, é criado um processo para atender às requisições deste cliente. Esta abordagem possibilita que o processo pai se preocupe apenas em receber as solicitações de conexão vindas dos diversos clientes, e repassar a responsabilidade de responder às requisições ao processo filho gerado por ele após o estabelecimento de cada conexão.

Com este modelo de implementação pode haver mais de uma conexão ativa simultaneamente entre o servidor e seus clientes. Além disso não existe conflito durante a escrita no socket, uma vez que são processos diferentes que o estão utilizando e tais processos não são executados simultaneamente no processador.

2.5 Servidor utilizando Threads e Fila

Nesta implementação, as requisições de cada cliente são atendidas e respondidas por um número fixo de threads. Após receber uma nova requisição a linha de execução principal do programa cria uma thread para atender a requisição solicitada pelo cliente que está ativo.

Por se tratar de um mesmo processo, para que não houvesse conflito durante a escrita/escuta no socket entre as threads ativas e o ramo de execução principal do programa, foi necessária a utilização da função *pthread_join*. A função *pthread_join* faz com que a thread principal espere que a thread passada como parâmetro para a função termine sua execução antes de prosseguir. Por causa desta necessidade de evitar conflitos entre as threads houve uma queda no desempenho do servidor.

2.6 Servidor utilizando Select

Esta implementação utiliza a função Select para criar uma estrutura que monitora inúmeros sockets abertos, o processo somente é acionado quando novos dados chegam em alguma das conexões monitoradas.

3.Principais Rotinas

As principais rotinas implementadas neste trabalho estão destacadas na Tabela 1.

Tabela 1

Função	**Descrição**
configura_porta	Esta função permite que o usuário defina as porta de execução do servidor.
menu	Esta função permite selecionar qual o modelo de servidor que será executado.
funcao	Esta função é utilizada pela thread para auxiliar na resposta de requisições dos clientes.
Iterativo	Esta função implementa o modo iterativo do servidor web.
Forked	Esta função implementa o modelo de servidor com multi processos utilizando fork.
Fila	Esta função implementa o modelo de servidor utilizando fila de tarefas.
Concorrente	Esta função implementa o modelo de servidor concorrente utilizando select.
resposta	Esta função tem como objetivo responder as requisições dos clientes enviando uma mensagem padrão de resposta.

4.Testes e Análise de Resultados

Os testes foram realizados utilizando um número fixo de 150 clientes enviando requisições para o respectivo servidor durante um tempo de 30 segundos. Para cada servidor foram feitos dez testes, e posteriormente foi calculada a média de requisições por segundo que cada um obteve.

4.1 Resultados Obtidos

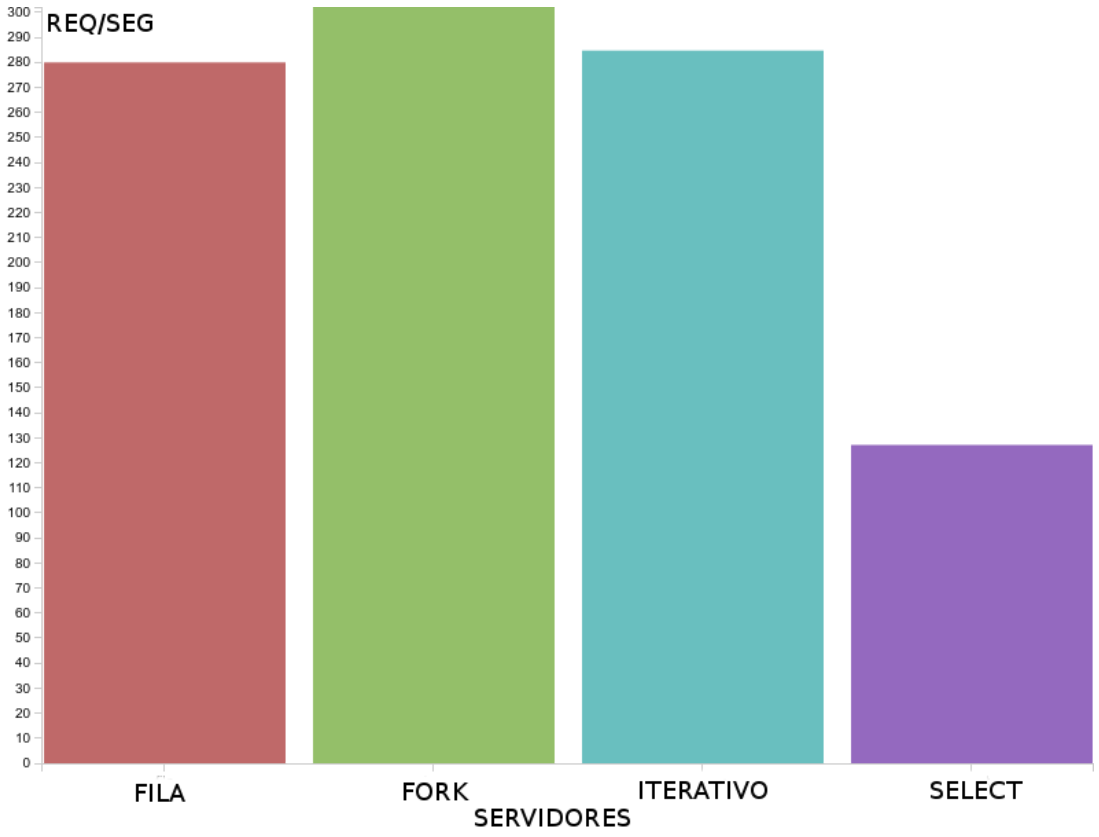
Os resultados obtidos após os testes estão apresentados na tabela 2 e gráfico 1 abaixo.

Tabela 2

Cliente HTTP: Siege
Número de usuários: 150
Tempo de execução: 30 segundos.

Nome	**Requisições por segundo**	**Falhas**
Fork	302.13	Não
Iterativo	284.718	Não
Fila	279.99	Não
Select	127.17	Sim

Gráfico 1



4.2 Análise de Resultados

Por meio dos testes realizados, é possível observar que o servidor utilizando *fork* foi o que obteve o melhor desempenho. Essa vantagem se deu em virtude de que a técnica adotada por este modelo de servidor permitiu que mais de uma conexão estivesse ativa simultaneamente, o que fez com que mais requisições pudessem ser atendidas ao final do período do teste.

Os modelos *iterativo* e *fila* obtiveram desempenhos muito semelhantes. Esse comportamento se deve porque na prática, mesmo utilizando técnicas diferentes para atendimento de requisições, os servidores se comportam de forma parecida pelo fato do servidor com fila ter que utilizar a função pthread_join para evitar conflitos entre suas diversas threads ativas na hora de escrever no socket.

O modelo implementado utilizando a função *select* apresentou um desempenho bastante inferior aos demais métodos apresentados. Essa baixa performance final se deve ao fato de erros na implementação, o que resultou em um número excessivo de falhas e uma baixa capacidade de resposta.

5.Conclusão

Com a implementação de diferentes estratégias de servidores web foi possível identificar as dificuldades e particularidades de cada abordagem. Além disso foi possível aperfeiçoar os conhecimentos sobre os protocolos da camada de aplicação e de transporte, bem como observar como é o desempenho de cada um dos modelos de servidor implementados.

Além dos benefícios já destacados, a implementação desse trabalho possibilitou vivenciar a construção de um servidor web que é uma importante ferramenta da camada de aplicação e da web.

6.Referências Bibliográficas

Kurose, James F., e Keith W. Ross. *Redes de computadores e a internet: uma abordagem top-down*. N.p., 2010. Print.