Programação de Sistemas

MEEC – 2013/2014 Projecto – Funcionamento

Nº de Grupo (FENIX): **2**

Nº: **73099** Nome: **Maria Margarida Dias dos Reis** Nota prevista: **20** Nota: ____

Nº: **73373** Nome: **Ricardo Filipe Amendoeira** Nota prevista: **20** Nota: ____

Grau de participação no projecto de cada elemento do grupo (soma igual a 100%)?

Num: **73099** (**50 %**) Num: **73373** (**50 %**)

Funcionamento	**Não foi codificada**	**Funcionalidade codificada**			
Funcionalidades		**Não Funciona**	**Não sabe se funciona**	**Funciona c/ falhas**	**Funciona**
Verificação formato do pedido					✓
• Pedido inválido (400)					✓
Tratamento de pedidos					✓
• Ficheiros .html/.png (200)					✓
• Ficheiros de tipo incompatível (415)					✓
• Ficheiros inexistentes (404)					✓
Listagem de directorias					✓
• Com links					✓
Gerador de relatórios					✓
• /estatisticas/Pedidos					✓
• /estatisticas/ClearAll					✓
CGI					✓
• Execução					✓
• Envio da resposta					✓
• Terminação ao fim de 5 segundos					✓
Ficheiro de configurações (www.config)					✓
• Carregamento no início					✓
• Carregamento com o SIGUSR1					✓
• DOCUMENT_ROOT					✓
• CGI_ROOT (403)					✓
Terminação do servidor (CTRL-C / SIGTERM)					✓
Paralelismo		☑ Threads ☑ Processos			
Vários ficheiros .h e .c?					✓
Comentários no código fonte					✓
Makefile					✓
Compila sem erros e avisos					✓

Compila sem erros e avisos (opção **-Wall**)					✓
Testes de carga? (usando o comando **ab**)					✓

Recepção de um pedido

Quando chega um pedido de um determinado cliente ao servidor, a *thread* principal chama uma função responsável por fazer o *accept*. Esta função devolve para o processo principal o *file descriptor* da *socket* desse cliente, que é depois escrito numa FIFO, em conjunto com o endereço IP do cliente, a data e a hora em que o pedido foi aceite e o início da contagem em milisegundos da duração da resposta ao pedido do cliente.

A *pool* de *threads* que é criada ao início tem a responsabilidade de ler da FIFO a estrutura enunciada anteriormente e chamar uma função que analisa o pedido feito. Quando um determinado pedido já acabou de ser atendido o *file descriptor* da *socket* desse cliente fica livre e será reutilizado futuramente.

Threads e processos

São usados processos extra para executar outros programas, como o *ls* para listar as directorias ou os CGIs. Há 3 tipos diferentes de *threads* – uma *thread* que armazena as estatísticas numa lista, uma *thread* de controlo que cria mais threads de atendimento, quando necessário, e essas mesmas *threads* de atendimento a clientes.

Gestão de *threads* e processos

No início do programa é criada uma *pool* base de *threads* com 150 *threads*, sendo que este valor não se mantém constante ao longo do programa, sendo de facto crescente monotónico. É definido que o número de *threads* livres nunca pode cair abaixo de 10 e assim, quando há menos de 10 *threads* livres de atendimento aos clientes existe uma função de controlo e gestão de *threads* que cria o número necessário das mesmas para que voltem a ser 10. A ideia é, depois de criada a *pool* base criarem-se mais blocos de *threads*.

As *threads* são terminadas quando se interrompe o servidor, com a excepção da *thread* de controlo, que termina se já não for possível abrir novas *threads*.

Existem *threads* de atendimento aos clientes, uma *thread* que armazena as estatísticas e uma *thread* de controlo que cria novas *threads* de atendimento, quando necessário.

Controlo de tempo dos CGIs

Para a execução de CGIs é feito inicialmente um *fork*. O processo filho é responsável por executar o CGI e escrever o *output* deste na *socket* do cliente que efectuou o pedido. O processo pai começa por registar o tempo actual e entra depois num ciclo em que espera que o processo filho mude de estado dentro de 5 segundos. Se depois do ciclo se verifica que o processo filho não

mudou de estado então é porque nunca se completou a execução do CGI dentro dos 5 segundos e mata-se o processo filho, ou seja, interrompe-se a execução do CGI.

Armazenamento dos pedidos

Para armazenar os pedidos recorrem-se a diferentes estruturas, uma para armazenar as informações do pedido e uma para armazenar as informações relevantes para as estatísticas.

A primeira contém *file descriptor* da *socket* do cliente, o endereço IP do cliente, a data e hora em que o pedido foi aceite e a hora (com precisão de milissegundos) em que o pedido foi aceite, para contar o tempo de processamento do pedido e envio da resposta. Os dados são preenchidos nesta estrutura aquando do *accept*. Garante-se sincronização no acesso a estes dados pela FIFO para onde se escreve e de onde se lê esta estrutura, uma vez que um *named pipe* permite escritas e leituras atómicas.

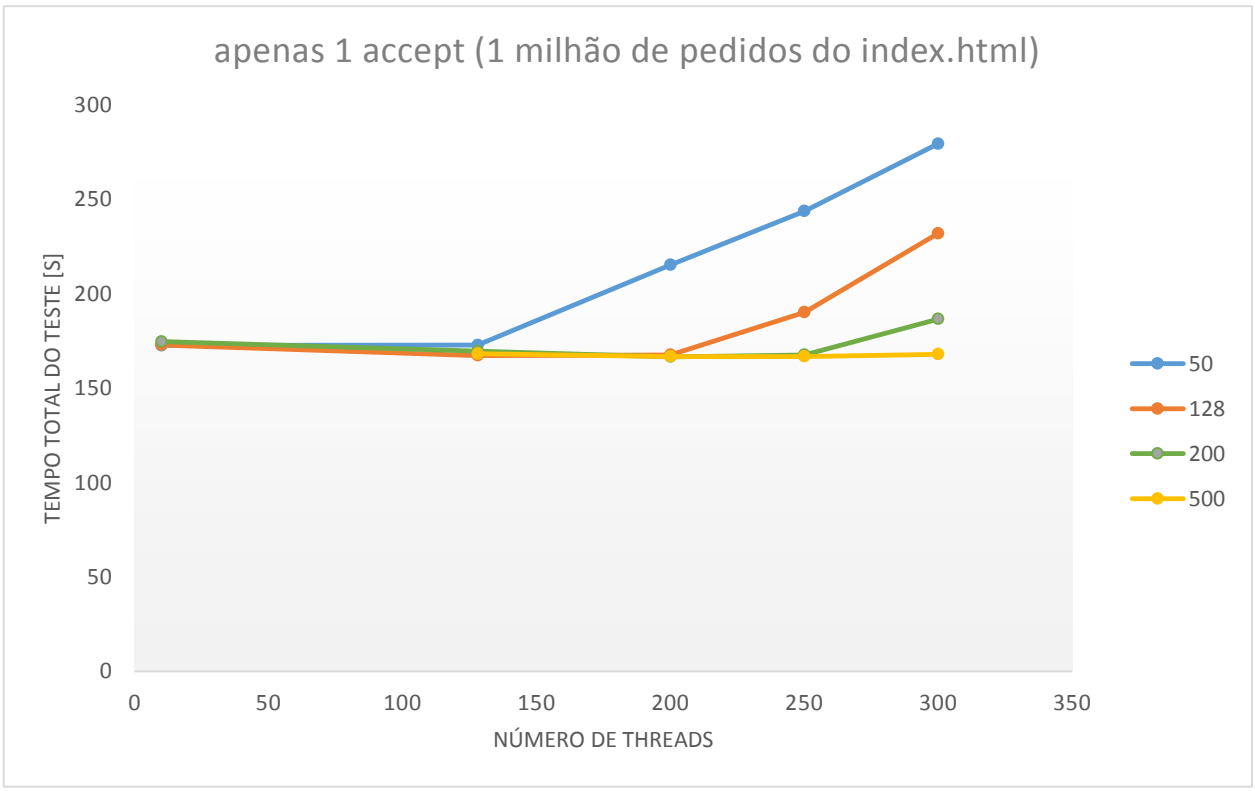
A estrutura das estatísticas contém a data e a hora em que o pedido foi aceite, qual o pedido feito, o código (200, 400, 404, 415 ou 403) do pedido, o endereço IP do cliente e o tempo de processamento e execução do pedido. Os dados são preenchidos nesta estrutura aquando da análise do pedido que foi feito, ou seja, uma vez que se analisou o tipo de pedido e quanto tempo demorou a sua resposta. Garante-se sincronização no acesso a estes dados pela FIFO para onde se escreve e de onde se lê esta estrutura, uma vez que um *named pipe* permite escritas e leituras atómicas.

Testes de carga

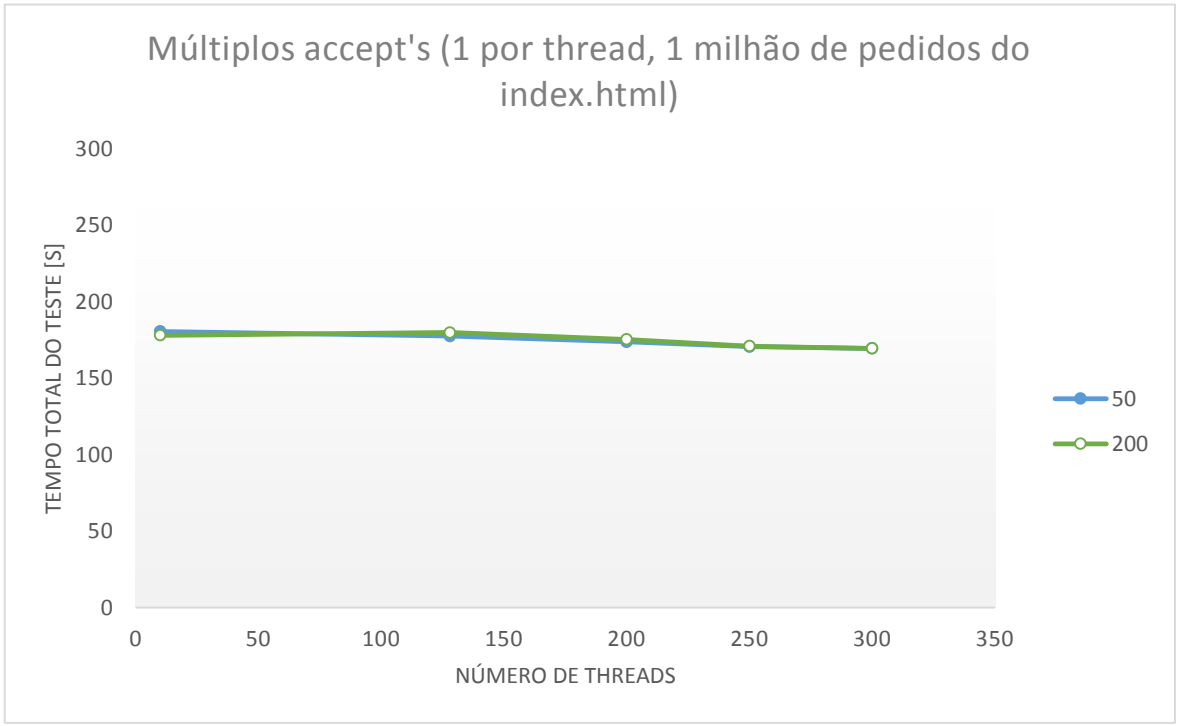
Foram feitos diversos testes com o *ab*, não só para garantir a estabilidade do servidor mas também para descobrir a melhor forma de controlar a criação de *threads* e a forma como é feito o *accept*.

Os gráficos seguintes são resultado de testes executados com o *ab* a correr na mesma máquina que o servidor com o objectivo de estudar diferentes implementações. Também se fizeram testes com o *ab* numa máquina diferente da do servidor mas esses resultados não foram apontados por serem apenas para garantir estabilidade.

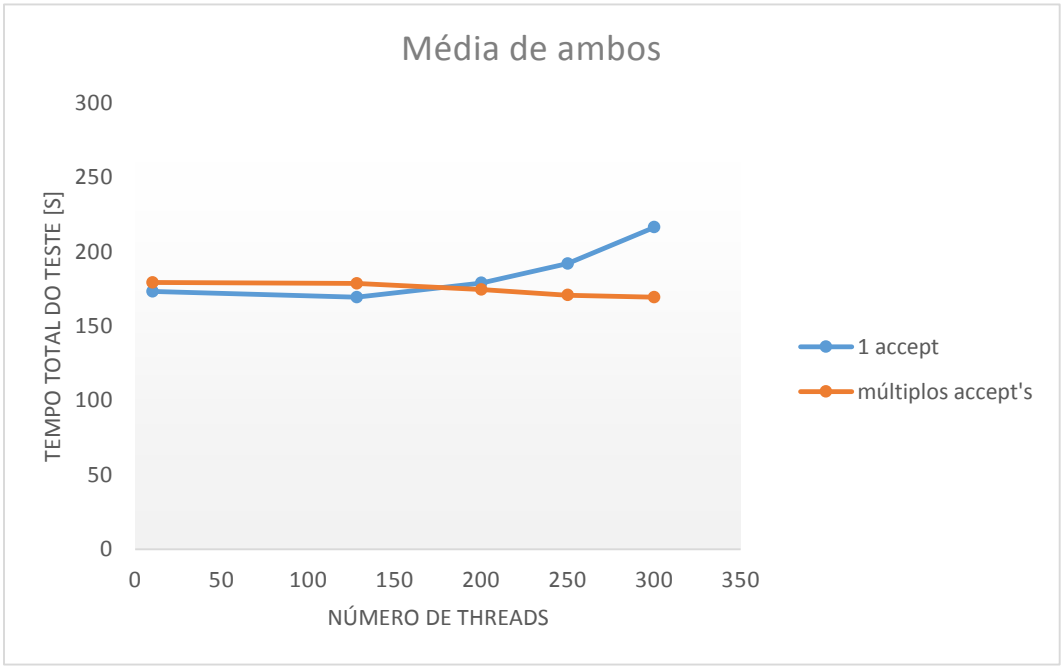
O seguinte gráfico compara a performance do servidor com um número fixo de *threads* e a responder a diferentes quantidades de pedidos concorrentes. Permitiu concluir que ter *threads* em excesso deteriora significativamente a rapidez das respostas.



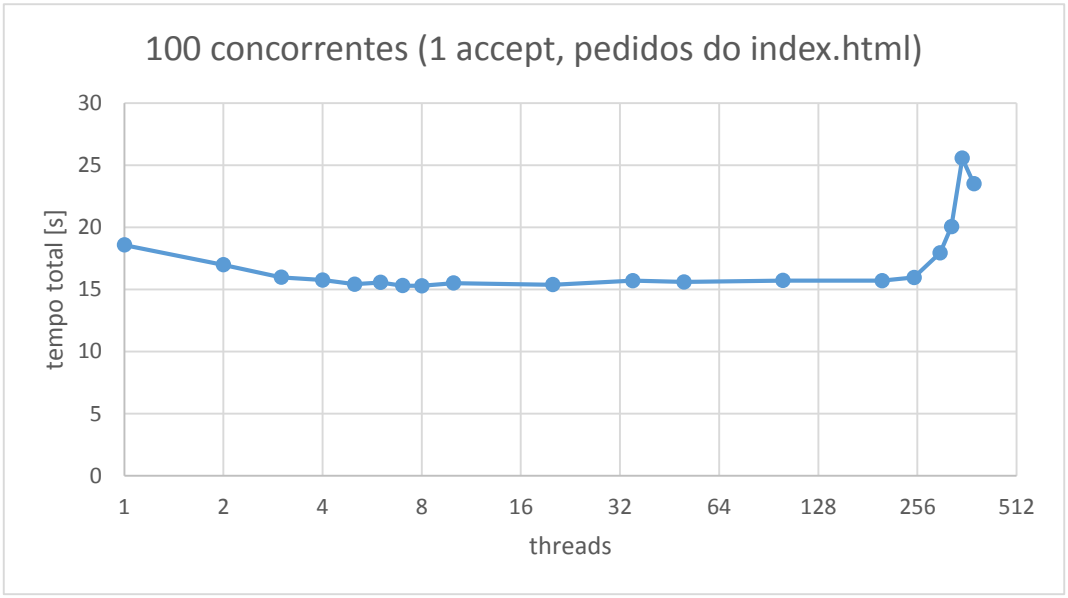
O gráfico seguinte mostra que com múltiplos *accepts* não há impacto negativo na performance do servidor se houver *threads* em excesso. No entanto, quando comparado ao método do gráfico anterior este método é ligeiramente mais lento (se o número de *threads* usado para o método anterior for adequado). Assim, optou-se por apenas 1 *accept.*



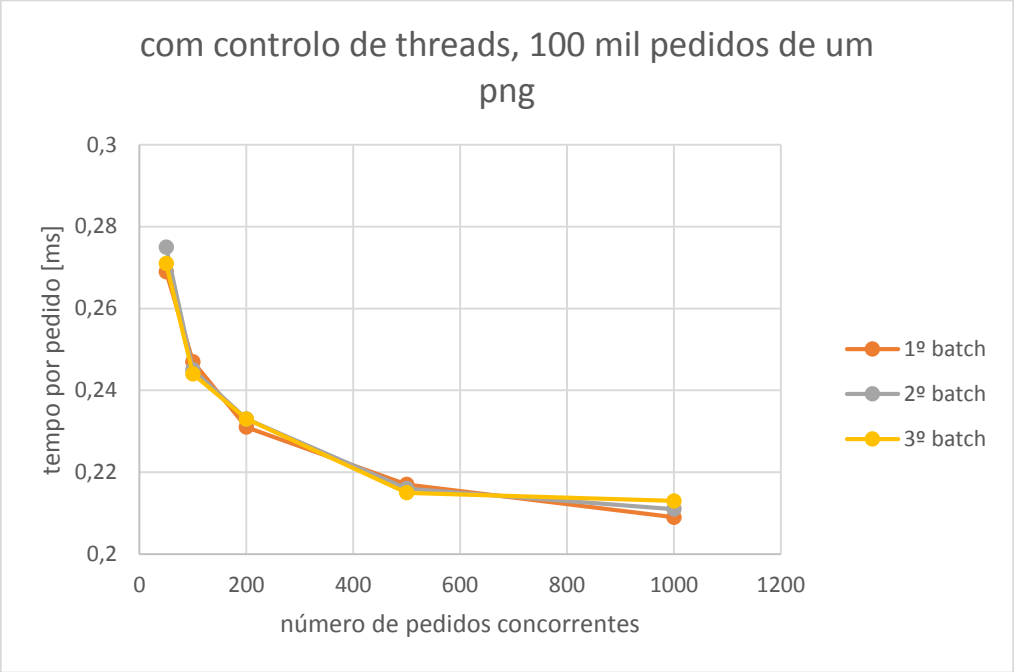
No gráfico abaixo é possível comparar mais facilmente ambos os métodos:



Nos testes do gráfico abaixo comparou-se o efeito do número de *threads* na rapidez das respostas.



Estes testes permitiram chegar ao método usado na versão final do servidor, com os seguintes resultados:



Para concluir, a versão final do servidor foi submetida a diversos testes de carga, com vários pedidos diferentes de forma a testar diversas funcionalidades (enviar ficheiros *html*, listar uma directoria, pedir a lista de estatísticas, etc…).

Estes testes tiveram entre 100 mil e 1 milhão de pedidos totais cada, com diferentes níveis de concorrência (entre 50 e 1000) e o servidor manteve-se estável, Porém, com um número muito elevado de pedidos concorrentes, para cima de aproximadamente 500 e com o *ab* a correr na mesma máquina que o servidor, surgem ocasionais erros 104.