Sistemas Operativos 2014/2015

Trabalho Prático

Servidor HTTP

1. Objetivos do trabalho

- Desenvolver um servidor *web* na linguagem de programação C com suporte de *multi-threading*, acesso a informação estática e dinâmica, gestão de configurações e de estatísticas.

- Explorar os mecanismos de gestão de processos, comunicação e sincronização entre processos no sistema operativo Linux.

2. Visão geral do funcionamento do servidor

A Figura 1 apresenta uma visão geral do funcionamento do servidor a desenvolver.

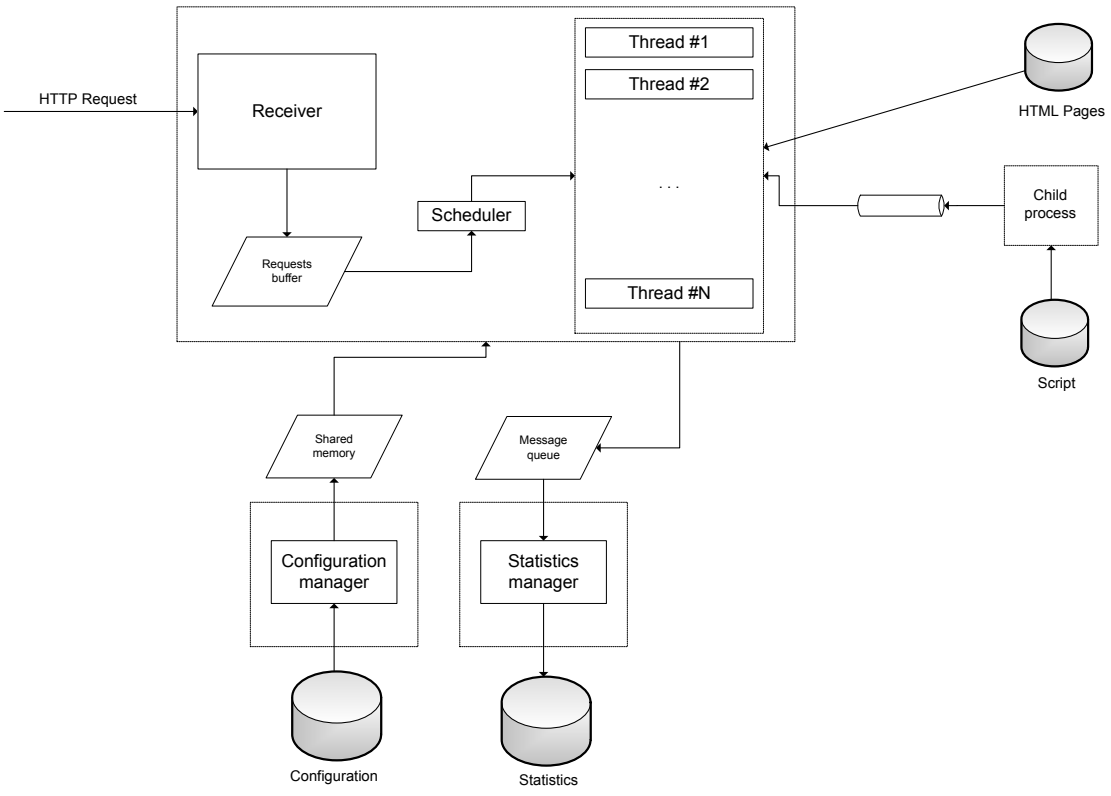

Figura 1 - Visão geral do funcionamento do Servidor

O servidor HTTP a implementar utiliza três processos, tal como a Figura 1 ilustra. Estes processos são responsáveis pelas seguintes funcionalidades:

- O processo principal (do ponto de vista das funcionalidades asseguradas) é responsável pela aceitação de novas ligações HTTP e pelo escalonamento de *threads* ou processos filho para tratamento dos pedidos de acesso a páginas ou execução de *scripts*.

- O processo de gestão de configurações é responsável pela leitura dos parâmetros de configuração do servidor a partir de um ficheiro de configuração. A informação de configuração determina aspetos de funcionamento do servidor e ficará acessível ao processo principal.

- O processo de gestão de estatísticas recebe do processo principal informação estatística acerca do funcionamento do servidor, em particular sobre as páginas acedidas ou *scripts* executados, e atualiza um ficheiro de *logs* (registos) com essa informação.

As funcionalidades a implementar encontram-se descritas em maior detalhe nas secções seguintes deste Enunciado.

3. Funcionalidades a implementar

3.1. Recepção de pedidos

Tal como a Figura 1 ilustra, o processo principal é responsável por receber as ligações HTTP e escalonar o tratamento dos pedidos efetuados pelos clientes (por exemplo *browsers*). Este processo suporta acessos HTTP dos seguintes tipos:

- Acessos a conteúdo estático (páginas HTML armazenadas em ficheiros), que são assegurados por *threads*.

- Acessos a conteúdos dinâmicos (resultado de execução de *shell scripts*), que são assegurados por processos filho do processo principal.

Após a recepção de uma nova ligação HTTP, o processo principal assegura as seguintes funcionalidades:

1. Interpretar o pedido efetuado pelo cliente (*browser*), para perceber se o acesso é a conteúdo estático ou dinâmico.

2. Colocar o pedido no *buffer* de pedidos, onde irá ser identificado pelo tipo de pedido, pelo ficheiro pretendido e pelo *socket* de comunicação com o cliente.

3. Um pedido de acesso a conteúdo estático (página HTML num ficheiro em disco) irá ser servido por uma das *threads* disponíveis, que tratará de ler o conteúdo do respetivo ficheiro e enviá-lo na forma de uma resposta HTTP ao cliente (*browser*) através do respetivo *socket*.

4. Um pedido de acesso a conteúdo dinâmico (resultado da execução de um *shell script*) deverá ser tratado por um processo filho do processo principal (criado pela *thread* responsável pelo pedido). Este processo deverá enviar o resultado da execução do *script* na forma de uma resposta HTTP ao cliente (*browser*) através do respetivo *socket*.

3.2. Tratamento de pedidos

Tal como descrito anteriormente, os pedidos HTTP são servidos por *threads*. O processo principal deverá criar no arranque uma *thread* responsável pelo escalonamento, bem como uma *pool* de *threads* para servir pedidos. A *thread* de escalonamento (identificada por *Scheduler* na Figura 1) ficará responsável por escalonar o atendimento dos pedidos utilizando as *threads* disponíveis na *pool*. O número de *threads* a criar (a dimensão dessa *pool*) é definido na configuração do servidor, sendo que o *buffer* de pedidos deverá ter capacidade para armazenar o dobro desse número. Caso num determinado instante não haja capacidade para armazenar no *buffer* um novo pedido o servidor deverá devolver ao cliente HTTP uma mensagem de erro apropriada.

Considera-se que a *thread* de escalonamento é responsável por validar o pedido, ou seja, verificar se o ficheiro em causa existe e, no caso de se tratar de um *script*, se a execução do mesmo está autorizada na configuração do servidor. O tratamento de um novo pedido por uma *thread* (a pedido da *thread* de escalonamento) deverá ser efetuado da seguinte forma.

- Caso se trate de um acesso a conteúdo estático (página HTML) a *thread* deverá abrir o ficheiro em causa, ler o seu conteúdo e devolvê-lo ao cliente (*browser*) na forma de uma resposta HTTP.

- Caso se trate de um acesso a conteúdo dinâmico a *thread* deverá criar um processo filho para execução do *script*, receber o resultado através de um *pipe* (criado para o processo filho em particular) e devolvê-lo ao cliente na forma de uma resposta HTTP. Os *scripts* passiveis de serem executados são definidos na configuração do servidor.

A *thread* de escalonamento deverá decidir qual é o próximo pedido a atender de acordo com diferentes políticas de escalonamento, definidas na configuração do servidor:

- Escalonamento FIFO (*First in First out*), em que o próximo pedido a atender será o mais antigo no *buffer* de pedidos.

- Escalonamento com prioridade a conteúdo estático, em que o próximo pedido a atender será o pedido de conteúdo estático mais antigo no *buffer* (independentemente da existência de pedidos para conteúdo dinâmico).

- Escalonamento com prioridade a conteúdo dinâmico, em que o próximo pedido a atender será o pedido de conteúdo dinâmico mais antigo no *buffer* (independentemente da existência de pedidos para conteúdo estático).

3.3. Estatísticas de funcionamento

Pretende-se armazenar, em ficheiro, estatísticas relativas ao funcionamento do servidor. Tal como a Figura 1 ilustra, o processo de gestão das estatísticas é responsável por receber a informação sobre os pedidos aceites e processados pelo servidor através de uma fila de mensagens. Cada mensagem conterá informação sobre um pedido HTTP tratado pelo servidor, informação esta que deverá ser escrita no ficheiro de *logs* na forma de uma linha única.

Para cada acesso HTTP (a conteúdo estático ou dinâmico) o processo de gestão das estatísticas deverá armazenar no ficheiro de *logs* (numa única linha) a seguinte informação:

Tipo de pedido (estático ou dinâmico)
Ficheiro HTML lido ou o script executado
Número da thread na pool responsável por atender o pedido
Hora de recepção do pedido pela thread e hora de finalização (após ter enviado o resultado ao cliente)

Para além das estatísticas anteriores, o processo de gestão das estatísticas deverá ser capaz de enviar para a consola, após receber um sinal do tipo SIGHUP, a seguinte informação:

Hora de arranque do servidor e hora atual
Número total de acessos a conteúdo estático atendidos
Número total de acessos a conteúdo dinâmico atendidos
Número total de pedidos recusados

3.4. Parâmetros de configuração

Pretende-se dispor de uma forma de definir alguns parâmetros de configuração do servidor num ficheiro. Tal como a Figura 1 ilustra, o processo de gestão das configurações é responsável pela leitura das configurações definidos num ficheiro e pelo seu armazenamento numa zona de memória partilhada. O processo principal (bem como as várias *threads*) poderão consultar a informação de configuração sempre que necessário através dessa zona de memória partilhada.

A configuração é lida a partir do ficheiro de configuração para a memória partilhada durante o arranque do Servidor, ou em alternativa após a recepção, por parte do processo de gestão das configurações, de um sinal do tipo SIGHUP. Esta funcionalidade permitirá alterar a configuração do Servidor com o mesmo em funcionamento. O ficheiro de configuração deverá permitir definir os seguintes parâmetros de configuração do Servidor:

Porto para o servidor *Número de threads da pool* *Política de escalonamento de threads* *Lista de shell scripts autorizados, definidos pelo nome do ficheiro*

3.5. Arranque e terminação do Servidor

No seu arranque o servidor deverá criar os três processos (principal, gestão de estatísticas e gestão de configurações), bem como as *threads* e os vários recursos de comunicação e sincronização considerados necessários.

O Servidor deverá estar igualmente preparado para terminar, após a recepção, por parte do processo principal, de um sinal do tipo SIGINT. Nessa altura todos os processos devem ser terminados e deverá fazer-se a limpeza no sistema de todos os recursos partilhados.

4. Utilização do protocolo HTTP e código de exemplo fornecido

Pretende-se que o servidor a implementar assegure apenas algumas funcionalidades essenciais previstas no protocolo HTTP (*HyperText Transfer Protocol*). A programação das funcionalidades necessárias está ilustrada no código de exemplo fornecido com o Enunciado, em particular:

- Ativação de *sockets* para aceitação de ligações e comunicação com clientes.
- *Parsing* simples das mensagens HTTP para detecção do tipo de pedido.
- Devolução de códigos HTTP ao cliente em caso de erro, nomeadamente relativos à inexistência de ficheiros HTML ou impossibilidade de execução de *scripts*.

Para assegurar o acesso a conteúdos através de uma ligação HTTP é necessário começar por interpretar o cabeçalho do pedido recebido pelo Servidor, em particular o comando "GET". Iremos considerar que este comando é utilizado para ambos os tipos de acesso. A Figura 2 ilustra as comunicações entre um cliente (*browser*) e o servidor HTTP para acesso a uma página HTML (conteúdo estático). Tal como é visível na figura, a resposta enviado pelo Servidor ao cliente começa por conter um cabeçalho HTTP antes do código HTML da página pedida.

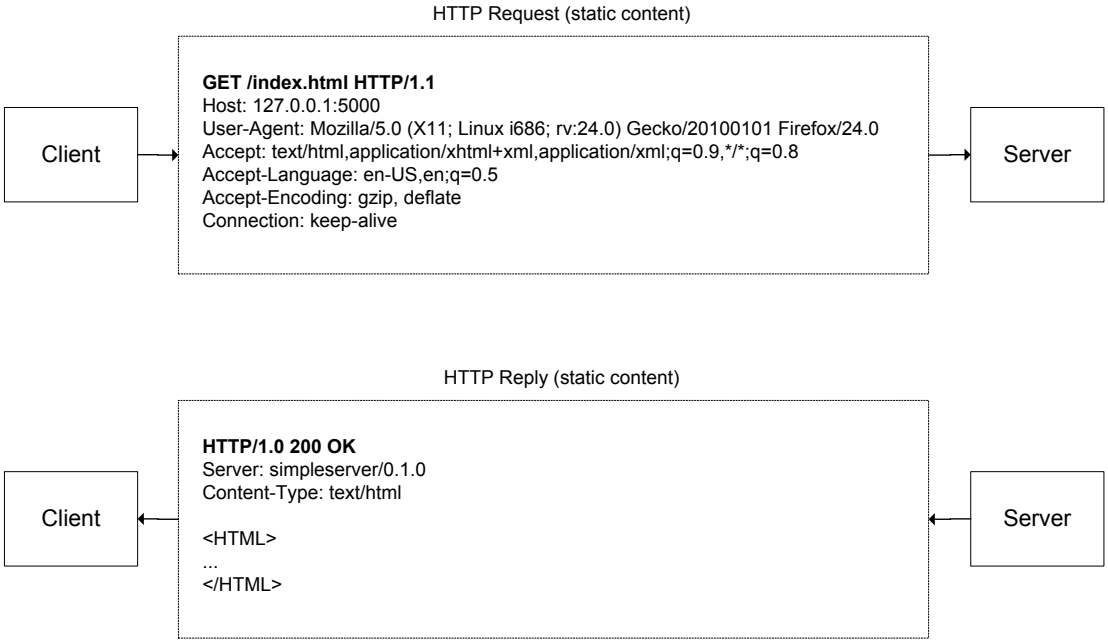

Figura 2 - Acesso a conteúdo estático através de HTTP

A Figura 3 ilustra as mesmas comunicações para execução remota de um *script* (acesso a conteúdo dinâmico).

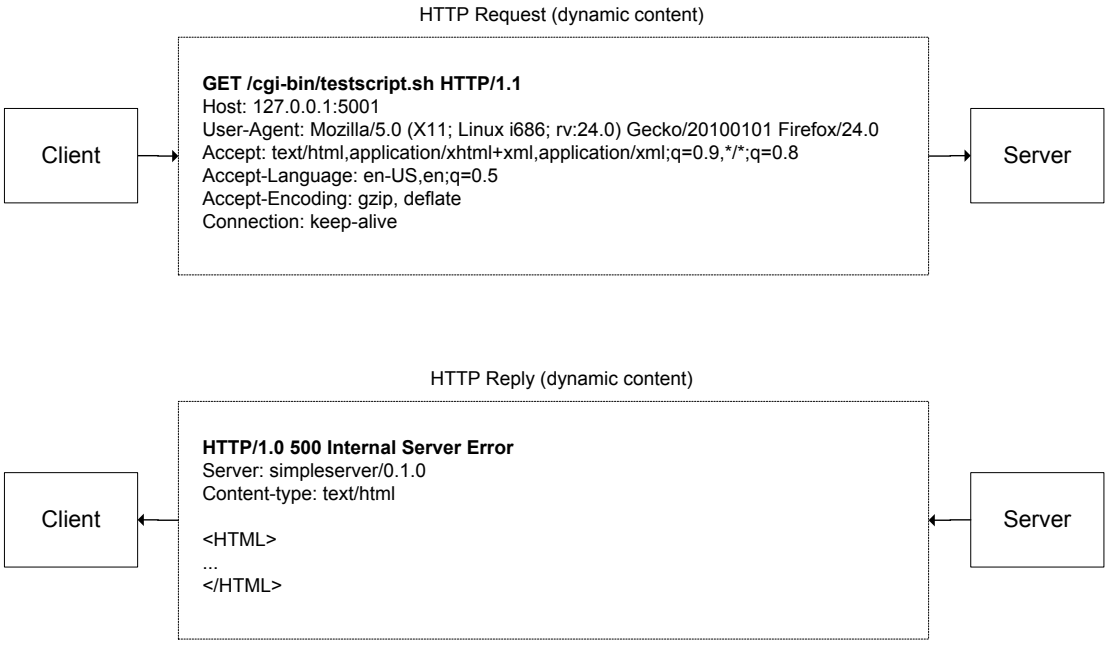

Figura 3 - Acesso a conteúdo dinâmico através de HTTP

No exemplo anterior, estamos a considerar que a execução do *script* é pedida utilizando igualmente o comando "GET" do HTTP, tal como previsto na especificação CGI (*Common Gateway Interface*), sem lugar à passagem de parâmetros. Neste caso consideramos igualmente que o *script* não se encontra autorizado para execução no servidor.

5. Checklist

Processo	Tarefa	Quantidade de trabalho
Servidor	Utilizar o `fork()` para criar processos filhos	5%
	Criar *pipes* "*unnamed*/*named*"	2%
	Ler informação do(s) *pipe* corretamente	10%
	Lançar *shell scripts*	5%
	Criar, inicializar e gerir a *pool* de *threads*	10%
	Colocar e retirar pedidos do *buffer* corretamente com recurso a semáforos, *mutexes* ou variáveis de condição para sincronização	15%
	Fornecer respostas corretas aos pedidos HTTP	2%
	Aplicar as configurações	5%
	Prevenir interrupções indesejada por sinais e fornecer a resposta adequada ao SIGINT	2%
	Criar e mapear a região de memória partilhada	2%
	Ler a configuração a partir da memória partilhada	5%
	Criar e inicializar a fila de mensagens	2%
	Escrever estatísticas corretamente para a fila de mensagens	5%
Configuração	Ler ficheiro de configuração	5%
	Escrever informação em memória partilhada	5%
	Prevenir interrupções indesejada por sinais e fornecer a resposta adequada ao SIGHUP	2%
Estatísticas	Ler dados da fila de mensagens	5%
	Criar o ficheiro para guardar as estatísticas	2%
	Escrever no ficheiro	5%
	Prevenir interrupções indesejada por sinais e fornecer a resposta adequada ao SIGHUP	2%
Geral	Detecção e tratamento de erros.	2%
	Terminação dos processos filhos quando o processo Servidor termina. Libertação de recursos e limpeza ao terminar a aplicação.	2%

Notas importantes

- **Não será tolerado plágio ou qualquer outro tipo de fraude**. Tentativas neste sentido resultarão na **classificação de ZERO VALORES** e na consequente **reprovação na cadeira**.
- Em vez de começar a programar de imediato pense com tempo no problema e estruture devidamente a sua solução.
- Inclua na sua solução o código necessário à detecção e correção de erros.
- Evite esperas ativas no código e assegure a terminação limpa do servidor, ou seja com todos os recursos utilizados a serem removidos.
- Utilize um *makefile* para simplificar o processo de compilação.
- Inclua informação de *debug* que facilite o acompanhamento da execução do programa, utilizando por exemplo a seguinte abordagem:

```
#define DEBUG //remove this line to remove debug messages
(...)
#ifdef DEBUG
printf("Creating shared memory\n");
#endif
```

6. Metas, entregas e datas

Data	Meta	
Semana de 24/11/2014	Demonstração intermédia	Os alunos deverão apresentar o seu trabalho nas aulas PL, preparando uma demostração do trabalho efetuado até ao momento, que deverá representar aproximadamente 25% do trabalho global necessário para terminar o projeto.
12/12/2014	Entrega final	O projeto final deverá ser submetido no InforEstudante, tendo em conta o seguinte: • Os nomes e números dos alunos do grupo devem ser colocados no início dos ficheiros com o código fonte. Inclua neste local também informação sobre o tempo total despendido (pelo dois elementos do grupo) no projeto. • Com o código deve ser entregue um relatório sucinto (no máximo 2 páginas A4) que explique as opções tomadas na construção da solução. • Crie um arquivo no formato ZIP com todos os ficheiros do trabalho.
Semana de 15/12/2014	Defesa	Defesas funcionais em grupo nas aulas PL desta semana. As defesas individuais escritas ocorrerão no dia 17/12/2014.