Sistemas Operativos 2016/2017

Trabalho Prático

Servidor HTTP

1. Objetivos do trabalho

- Desenvolver um servidor *web* na linguagem de programação C. Este servidor irá ser capaz de servir pedidos HTTP em simultâneo, quer a páginas estáticas existentes num ficheiro HTML, quer a páginas estáticas comprimidas, bem como gerir alguns aspetos de configuração e gestão de estatísticas.

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

2. Visão geral do funcionamento da aplicação

A Figura 1 apresenta uma visão geral do funcionamento do servidor a implementar, no contexto do Trabalho Prático.

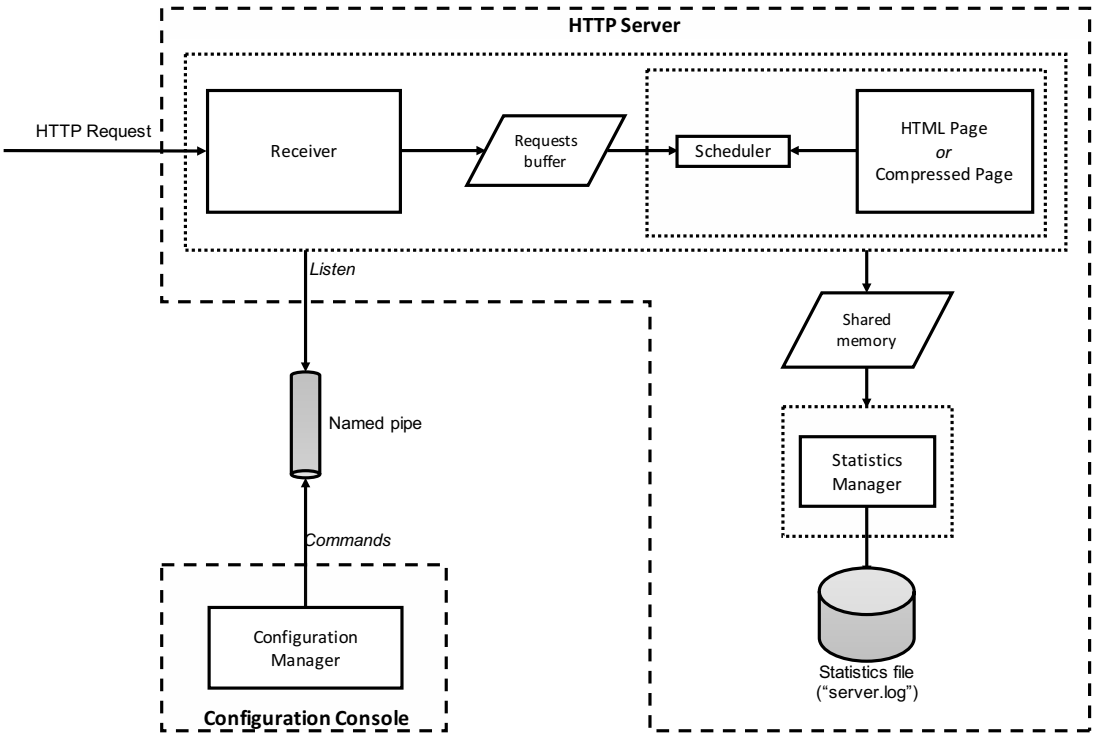

Figura 1 - Visão geral do funcionamento do Servidor

Tal como a figura anterior ilustra, o **servidor HTTP** a implementar utiliza dois processos. Estes processos são responsáveis pelas seguintes funcionalidades:

- O processo principal é responsável pela aceitação de novas ligações HTTP e pelo encaminhamento do pedido para o *scheduler*, que é responsável por todas as operações necessárias à resposta a pedidos dos clientes (pedidos HTTP a páginas ou conteúdo comprimido). As configurações iniciais do servidor são lidas a partir de um ficheiro e as subsequentes configurações são recebidas por um *named pipe*.

- O processo de gestão de estatísticas comunica por memória partilhada com o processo principal. A partir da informação de acessos a páginas enviada pelo processo principal, este processo escreve informação estatística agregada no écran (detalhada mais à frente), e a informação sobre os acessos para um ficheiro. O ficheiro de estatísticas deverá ser mapeado em memória (ser um MMF, *memory-mapped file*).

Um segundo executável implementa uma **consola de configuração** baseada em um único processo:

- O processo de gestão de configurações é um processo independente que comunica com o principal através de um *named pipe*. Este processo permite ao utilizador pode introduzir comandos simples quando desejar alterar a configuração do servidor. Para que a comunicação seja possível, o nome do *named pipe* e os comandos possíveis deverão ser conhecidos previamente pelos dois processos.

A seguir explicam-se em maior detalhe as funcionalidades a implementar na aplicação.

3. Descrição das funcionalidades a implementar

3.1. Receção 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. Pretende-se que a aplicação suporte acessos a dois tipos de recursos, através do Protocolo HTTP:

- A conteúdo estático, ou seja, a páginas HTML armazenadas em ficheiros.

- A conteúdo estático comprimido, ou seja, ficheiros alojados no servidor com extensão ".gz". Nesta situação, o servidor deverá descomprimir o ficheiro antes de enviar o seu conteúdo ao cliente.

Após a receção 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 normal ou comprimido.

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. Efetuar as operações necessárias para servir o pedido, enviando o respetivo resultado na forma de uma resposta HTTP ao cliente (*browser*), através do respetivo *socket*.

3.2. *Scheduler* e tratamento de pedidos

Tal como descrito anteriormente, o servidor deverá dispor da capacidade de responder a pedidos HTTP concorrentes, ou seja, de forma paralela. Para este efeito, pretende-se dispor de um *scheduler* que, de acordo com a política definida na configuração do servidor, controle a forma como tais pedidos são servidos. Na implementação desta componente deverá ter em conta os seguintes requisitos:

- Na implementação do *scheduler* o aluno deverá implementar a solução que achar mais adequada. O tratamento dos pedidos HTTP deverá ser realizado recorrendo a uma *pool* de *threads*.

- O *scheduler* é responsável por validar o pedido antes de escalonar a sua execução, ou seja, verificar se o ficheiro em causa existe e, no caso de se tratar de um ficheiro comprimido, se a sua descompressão e envio do respetivo conteúdo para o cliente se encontra autorizada na configuração do servidor.

- Deverá ser implementada uma solução que permita servir pedidos HTTP de forma eficiente e concorrente, ou seja, num determinado instante, o servidor deverá ser capaz de servir vários pedidos em simultâneo.

- De acordo com a capacidade do servidor, caso num determinado instante não haja capacidade para servir um novo pedido o servidor deverá devolver ao cliente HTTP uma mensagem de erro apropriada.

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

- Caso se trate de um acesso a conteúdo estático comprimido, o ficheiro deverá ser descomprimido, sendo o resultado da sua execução devolvido ao cliente, igualmente na forma de uma resposta HTTP.

- O *scheduler* deverá decidir qual é o próximo pedido a atender, de acordo com a política de escalonamento configurada no servidor, em particular uma das seguintes:

    - 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.*

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

3.3. Informação estatística sobre pedidos tratados pelo servidor

Pretendem-se armazenar, em ficheiro, estatísticas relativas ao funcionamento do servidor. Tal como a Figura 1 ilustra, o processo de gestão de estatísticas é responsável por receber informação sobre os pedidos aceites e tratados pelo servidor, através da memória partilhada.

Após a receção de um sinal do tipo SIGUSR1, o processo de gestão de estatísticas deverá escrever para o écran a seguinte informação estatística agregada:

- Número total de pedidos servidos (páginas estáticas).
- Número total de pedidos servidos (ficheiros comprimidos).
- Tempo médio para servir um pedido a conteúdo estático não comprimido.
- Tempo médio para servir um pedido a conteúdo estático comprimido.

Após a receção de um sinal do tipo SIGUSR2, o processo de gestão de estatísticas deverá fazer um *reset* à informação estatística agregada.

Como referido anteriormente, o processo de gestão de estatísticas escreve informação sobre todos os pedidos servidos pelo servidor, para um ficheiro mapeado em memória (MMF). Tal como indicado na Figura 1, este ficheiro deverá ter o nome "server.log". Neste ficheiro, o processo deverá colocar a seguinte informação sobre cada pedido (na forma de uma linha por cada pedido tratado):

Tipo de pedido (estático normal ou comprimido) *Ficheiro HTML lido, no formato normal ou comprimido* *Hora de receção do pedido* *Hora de terminação de servir o pedido (após envio do resultado ao cliente)*

3.4. Arranque e terminação do Servidor

No seu arranque, o servidor deverá criar os processos necessários (principal e de gestão de estatísticas), bem como todos os restantes recursos de comunicação e sincronização necessários. O PID de cada um dos processos deverá ser escrito no ecrã. De seguida deverá ler a configuração inicial do ficheiro "config.txt" que deverá conter os seguintes dados:

Porto para o servidor
Política de escalonamento no tratamento de pedidos a conteúdo normal ou comprimido
Número de threads a utilizar para tratamento de pedidos pelo servidor
Lista de ficheiros comprimidos autorizados (definidos pelo nome do ficheiro)

Exemplo de um ficheiro de configuração:

```
SERVERPORT=50000
SCHEDULING=NORMAL
THREADPOOL=5
ALLLOWED=a.gz;b.gz
```

O Servidor deverá estar igualmente preparado para terminar, após a receção, por parte do processo principal, de um sinal do tipo SIGINT. Nessa altura, o servidor deverá começar por aguardar a terminação de todos os pedidos pendentes, e só depois terminar os processos e fazer a limpeza no sistema de todos os recursos partilhados.

3.5. Consola de configuração

Pretende-se dispor de uma consola para alterar alguns parâmetros de configuração do servidor de forma dinâmica. Para isso, o processo de gestão de configurações será responsável por receber comandos do utilizador e pelo seu envio ao processo principal através de um *named pipe* (Figura 1).

A consola de configuração pode ser executada a qualquer momento, e o processo principal deve funcionar normalmente esteja a consola a ser executada ou não. Do lado do processo principal, este estará sempre à escuta do *named pipe*.

As mensagens a trocar entre os dois processos devem ser definidas antecipadamente de modo a que ambas as partes conheçam a sua estrutura e o seu significado. Deverá ser permitido alterar todos os parâmetros de configuração (os mesmos que estão presentes no "config.txt") excepto o número do porto do servidor. De notar que, caso seja necessário alterar o número de *threads* utilizadas pelo processo principal, todas as *threads* deverão terminar o pedido que estão a tratar, ser eliminadas e posteriormente criada uma nova *pool* com o número especificado, que deverá retomar a resolução de pedidos. Nota:

Soluções mais eficientes, que não impliquem recriar a totalidade da *pool*, poderão ser valorizadas.

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 deteção do tipo de pedido (a página HTML normal ou comprimida).
- Devolução de códigos HTTP ao cliente em caso de erro, nomeadamente relativos à inexistência de ficheiros com conteúdo estático normal ou comprimido.

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. Tal como é visível na figura, a resposta enviada pelo Servidor ao cliente começa por conter um cabeçalho HTTP antes do código HTML da página pedida.

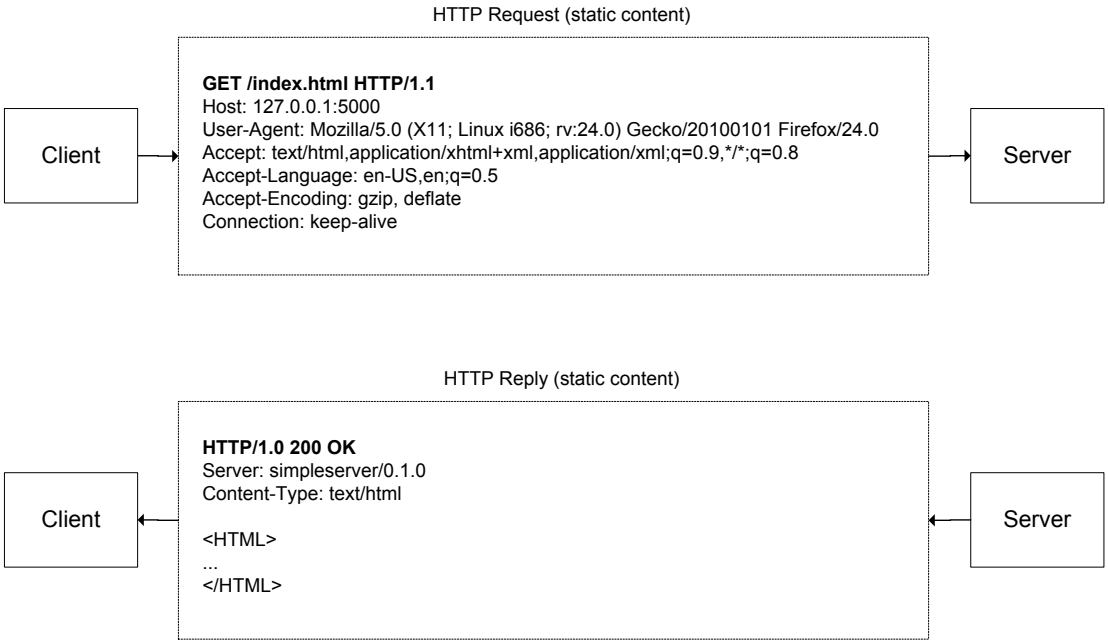

Figura 2 - Acesso a conteúdo estático através de HTTP

5. Checklist

Esta lista serve como indicadora das tarefas a realizar e assinala as componentes que serão objecto de avaliação na defesa intermédia.

Processo	**Tarefa**	**Avaliado na defesa intermédia?**
Servidor	Arranque do servidor e aplicação das configurações existentes no ficheiro "config.txt"	S
	Criação de todos os processo filho necessários	S
	Criação da memória partilhada	S
	Criação da *pool* de *threads*	S
	Criação do *named pipe*	
	Leitura correcta dos comandos recebidos pelo *named pipe*	
	Aplicação das configurações recebidas pelo *named pipe*	
	Suporte do *scheduler* e política de escalonamento	
	Suporte de concorrência no tratamento de pedidos	
	Leitura correcta dos pedidos feitos por HTTP e a sua colocação no *buffer* de pedidos	S
	Resposta a pedidos de páginas estáticas	S (apenas prova de conceito)
	Resposta a ficheiros comprimidos (extensão .gz)	
	Sincronização com mecanismos adequados (semáforos, *mutexes* ou variáveis de condição)	
	Prevenir interrupções indesejada por sinais e fornecer a resposta adequada aos vários sinais	
	Utilização da memória partilhada	
Configuração	Ler comandos do utilizador	
	Enviar comandos através do *named pipe*.	
Estatísticas	Ler informação da memória partilhada	
	Escrever a informação estatística no ficheiro mapeado em memória ("server.log")	
	Enviar informação agregada para o écran	
	Fazer *reset* a estatísticas agregadas	
Geral	Deteção e tratamento de erros.	S
	Terminação dos processos filho quando o processo principal termina. Libertação de recursos e limpeza ao terminar a aplicação.	S

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 deteção 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
```

- Todos os trabalhos devem funcionar na student2.dei.uc.pt ou, em alternativa, na VM fornecida.
- A defesa final do trabalho é obrigatória. A não comparência na defesa final implica a classificação de 0 valores no trabalho.
- O trabalho pode ser realizado em grupos de até 2 alunos.

6. Metas, entregas e datas

Data	Meta	
Semana de 24/10/2016	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. A defesa intermédia valerá 25% da cotação do projeto.
04/12/2016	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 (pelos 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. A defesa final valerá 75% da cotação do projecto.
05/12/2016 a 21/12/2016	Defesa	Defesas funcionais em grupo nas aulas PL.