FEUP Universidade do Porto
Faculdade de Engenharia

Cliente e Servidor HTTP

Redes de Computadores 2006/07

Relatório

Mestrado Integrado em Engenharia Informática e Computação

Faculdade de Engenharia da Universidade do Porto

02 de Junho de 2007

Cláudio Costa ei03032
Gonçalo Senra ei01091
Simão Castro ei04100

Índice

Introdução

O trabalho descrito neste relatório foi realizado no âmbito da disciplina de Redes de Computadores do Mestrado Integrado em Engenharia Informática e Computação (MIEIC) da Faculdade de Engenharia da Universidade do Porto (FEUP), com o intuito de desenvolver e estudar o protocolo de implementar um cliente e um servidor HTTP muito simples, utilizando C/C++ como linguagens de programação em ambiente UNIX.

Objectivos

- O cliente deverá ser capaz de enviar comandos simples HTTP, nomeadamente GET e GET condicional.
- O cliente deverá ser capaz de comunicar directamente com qualquer servidor HTTP.
- O servidor deverá responder a comandos GET e GET condicional.
- O servidor deverá ser capaz de comunicar directamente com qualquer cliente HTTP.
- O servidor deverá ser capaz de comunicar com mais do que um cliente simultaneamente.

Melhoramentos:

- Capacidade do cliente comunicar com *proxies* HTTP.
- Capacidade do cliente descarregar uma página *web* completa.
- Utilização de ligações persistentes.

Desenvolvimento

No desenvolvimento deste trabalho criaram-se dois ficheiros-fonte, respectivamente para o programa cliente e servidor: cliente.c e servidor.c

Para cada um deles fazemos agora uma explicação das funções principais.

Cliente

```
int main(int argc, char* argv[]);
```

A função *main* é a função principal do programa. Aqui é mostrado o menu principal (utilizando a função respectiva) e executada a opção escolhida.

```
int getFicheiro(int sockfd, char* ficheiro);
```

Esta função serve para transferir o ficheiro pedido, através da utilização de *sockets*, identificando os *headers* do servidor (utilizando as funções descritas na próxima subsecção).

```
int ver_site(char* url, int condicional);
```

A função *ver_site* é utilizada para as opções 1 e 2 do menu, permitindo consultar um *website*, através do comando GET (condicional ou não). Aqui é criado o socket para fazer a ligação, criado o sistema de directórios do *website*, e executado o comando GET. Por fim é descarregado o ficheiro, através da função anterior.

CACHE

```
int fill_cache(FILE* file);
```

A função *fill_cache* tem como objectivo ler o ficheiro de *cache* passado como argumento, preenchendo-o depois com os endereços.

```
int save_cache(FILE* file);
```

A função *save_cache* serve para registar um endereço no ficheiro de *cache*.

HEADERS

```
int check_length(char **headers,int nrheaders);
```

A função *check_length* tem como finalidade devolver o valor referente ao *header* Content-Length, com o tamanho da página a ler.

```
char * check_modified(char **headers,int nrheaders);
```

Esta função é semelhante à anterior, com a diferença que vai tentar ler o valor do *header* Last-Modified. Este *header* contém a informação relativa à data da última alteração do ficheiro, sendo essa data (*timestamp*) devolvida pela função.

```
int check_errors(char*headers,int nrheaders);
```

Com esta função pretende-se identificar os erros devolvidos pelo servidor.

```
int check_chunked(char ** headers,int nrheaders)
```

Esta função tem como função detectar a existência do *header* Transfer-Encoding, detectando o modo como de codificação da tranferência. Se o valor for "chunked" a função retorna 1, caso contrário retorna 0.

```
void get_chunked(int filled, char* cenas, int sockfd)
```

Esta função permite receber e tratar páginas que tenham o *header* Transfer-Encoding com o valor "chunked". Isto significa que a informação é enviada em blocos de dados, sendo devidamente tratada nesta função.

MENU

```
int menu();
```

A função *menu* serve para mostrar no ecrã as opções disponíveis. É a partir desta função que o nosso programa se desenrola. Para tal o nosso menu apresenta seis hipóteses que o utilizador poderá seleccionar.

Passamos a explicar com maior detalhe cada uma dessas hipóteses:

1- Visitar um *website*: é pedido ao utilizador que insira a página que pretende visitar, sendo esta transferida para o computador, através da instrução GET.

2- Visitar um *website*, se tiver sido actualizado: semelhante à opção anterior, com a diferença que a página só é descarregada se tiver sido actualizada desde a última visita. Esta instrução é conhecida como GET condicional, utilizando-se para essa verificação o *request header* If-Modified-Since.

3- Visitar um *website* e descarregar todo o seu conteúdo [não implementado]: esta funcionalidade não foi implementada.

4- Visitar um *website* via *proxy server* [não implementado]: esta funcionalidade não foi implementada.

5- Alterar a porta a utilizar nas ligações: permite a alteração da porta utilizada nas ligações a efectuar. Por defeito, a porta utilizada é a nº 80.

6- Sair: Esta opção termina a execução da aplicação.

Servidor

```
int getFileSize(int fd)
```

A função *getFileSize* serve para devolver o tamanho do ficheiro, com o file descriptor (fd) passado como parâmetro.

```
int check_host(char**headers,int numero)
```

A função *check_host* é utilizada para identificar o valor do *header* Host. Este *header* tem informação relativa ao *host* da página visitada.

```
int handle_request(int sockfd,char*request)
```

A função *handle_request* detecta o pedido do cliente, tratando-o devidamente. A função devolve 1 no caso de executar correctamente o pedido e -1 se receber um *bad request*.

```
int main(int argc, char**argv)
```

A função *main* cria a ligação, através da utilização de *sockets*, ficando depois a aguardar um pedido de algum cliente. Assim que este pedido é recebido é criado um novo processo (através da instrução *fork()*), onde esse pedido será tratado pela função *handle_request*.

Conclusões

Os objectivos do trabalho foram praticamente atingidos, tendo sido desenvolvidas as aplicações cliente e servidor HTTP, como era pedido no enunciado. Apenas não conseguimos realizar os melhoramentos, por insuficiência de tempo, face à data de entrega.

Com o desenvolvimento deste trabalho, foi possível testar na prática o protocolo HTTP, bem como assimilar alguns conhecimentos básicos sobre o funcionamento de uma rede de computadores.

Manual de Utilização

Compilação

A aplicação cliente e a aplicação servidor foram feitas separadamente, tendo diferentes *makefiles*. No entanto as suas *makefiles* têm o mesmo funcionamento.

Para compilar o ficheiro-fonte, basta executar a instrução

```
> make
```

Para eliminar os *object files* gerados basta executar a instrução

```
> make clean
```

Executar Cliente

Para se poder executar o cliente basta utilizar o seguinte comando:

```
> ./cliente
```

Executar Servidor

Para se poder executar o servidor, utiliza-se um comando semelhante.

```
> ./servidor
```

Referências

Literárias

- Comer, Douglas E.; Internetworking with TCP/IP
- Tanenbaum, Andrew S.; Computer Networks

Electrónicas

- Apontamentos das aulas teóricas
  http://paginas.fe.up.pt/~mleitao/RCOM/
- The Linux Serial Programming HOWTO, by Peter H. Baumann
  http://paginas.fe.up.pt/~jsc/RCOM/Serial-Programming-HOWTO
- Hypertext Transfer Protocol -- HTTP/1.1
  http://www.w3.org/Protocols/rfc2616/rfc2616.html
- Linux Socket Programming
  http://www.cs.utah.edu/~swalton/listings/sockets/programs/
- HTTP Made Really Easy
  http://jmarshall.com/easy/http/

Código-fonte

cliente.c

```
#include <stdio.h>
#include <stdlib.h>

//handle files
#include <fcntl.h>

//use of mkdir
#include <sys/stat.h>

//Stat()
#include <unistd.h>

#include <string.h>

//Defines e cenas
#define MAXLEN 1024

//MakeArgv
#include "helper.h"

//Estrutura Global que guarda a informaÃÃo do servidor
//struct sockaddr_in server_addr;

#include "socket.h"



#include <time.h>


//Porta a ser utilizada:
int porta = 80;


typedef struct{
     char * path;
     char * lastmodified;
}cachefile;

cachefile cache[100];
int cached = 0;

typedef struct{
     char *wday;
     char *mday;
     char *month;
     char *time;
     char *year;
}timestamp;

int fill_cache(FILE* file)
{
     int i=0;
     int j=0;

     while((cache[i].path = (char *)malloc(1024)) &&
fgets(cache[i].path,35,file)!=NULL)
     {
```

```
        cache[i].lastmodified = (char *)malloc(1024);
        fgets(cache[i].lastmodified,35,file);
        i++;
        cached++;
    }
    for(i=0;i<cached;i++)
    {
        while(cache[i].path[j]!='\n')j++;
        cache[i].path[j]='\0';
        j=0;
        while(cache[i].lastmodified[j]!='\n')j++;
        cache[i].lastmodified[j] ='\0';
        j=0;
    }
    return 1;
}

int save_cache(FILE*file)
{
    int i=0;
    for(i;i<cached;i++)
    {
        fprintf(file,"%s\n",cache[i].path);
        fprintf(file,"%s\n",cache[i].lastmodified);
    }
    return 1;
}

int menu()
{
    char resposta[50];
    int i;
    printf("\n  ------------------------------------------------\n ");
    printf(" -- RCOM | Redes de Computadores // FEUP 2007 --\n ");
    printf(" --                                            --\n ");
    printf(" -- Cliente HTTP | ei01091.ei03032.ei04100    --\n ");
    printf(" ------------------------------------------------\n ");
    printf("\n");
    printf("\nTem disponiveis as seguintes opcoes:\n");
    printf("1- Visitar um website\n");
    printf("2- Visitar um website, se tiver sido actualizado\n");
    printf("3- Visitar um website e descarregar todo o seu conteudo
[Nao implementado]\n");
    printf("4- Visitar um website via proxy server [Nao
implementado]\n");
    printf("5- Alterar a porta a utilizar nas ligacoes [Porta
definida: %d]\n",porta);
    printf("6- Sair\n> ");
    fgets(resposta,sizeof(resposta),stdin);
    i =0;
    while(resposta[i]!='\n')
        i++;
    resposta[i]='\0';
    return atoi(resposta);
}

char * check_modified(char **headers,int nrheaders)
{
    int i;
    char * aux;
    for(i=0;i<nrheaders;i++)
```

```
    {
        if(strncmp(headers[i],"Last-Modified:",strlen("Last-
Modified:"))==0)
        {
            strtok(headers[i]," ");
            aux = strtok(NULL,"\r\n");
            return aux;
        }
    }
    return "No Info";
}
/**
 *   200 OK - return 2
 *   3xx Redirection - return 3
 *   301 Moved Permanently - return 31
 *   304 Not Modified - return 34
 *   4xx Client Error - return 4
 *   400 Bad Request - return 40
 *   404 Not Found - return 44
 *   5xx Server Error - return 5
 *
 */
int check_errors(char*headers,int nrheaders)
{
    int i;
    char * aux;
    strtok(headers," ");
    aux = strtok(NULL," ");
    i = atoi(aux);
    switch(i)
    {
        case 200: return 2;
        case 301: return 31;
        case 304: return 34;
        case 400: return 40;
        case 404: return 44;
        default:break;
    }
    i = (int) i/100;
    return i;
}
int check_length(char **headers,int nrheaders)
{
    int i;
    for(i=0;i<nrheaders;i++)
    {
        if(strncmp(headers[i],"Content-Length: ",strlen("Content-
Length: "))==0)
        {
            strtok(headers[i]," ");
            return atoi(strtok(NULL,"\r\n"));
        }
    }
    return 0;
}

int check_chunked(char ** headers,int nrheaders)
{
    int i;
    for(i=0;i<nrheaders;i++)
    {
```

```
        if(strncmp(headers[i],"Transfer-Encoding:
chunked",strlen("Transfer-Encoding: chunked"))==0)
                return 1;
    }
    return 0;
}


void get_chunked(int filed,char * cenas, int sockfd)
{
    long length;
    int i,j=0;

    char byte;
    char * tamanho = (char*)malloc(sizeof(char)*500);
    int next=0;
    int k=0;
    char ** strip;
    int ntokens;

    ntokens = makeargv(cenas,"\r\n",&strip);
    length=strtol(strip[0],NULL,16);
    if(length ==0)
       return;
    for(i=1;i<ntokens;i++)
    {
        if(next)
        {
            length = strtol(strip[i],NULL,16);
            if(length ==0)
                return;
            next =0;
            k=0;
        }
        else
        {
            for(j=0;strip[i][j]!='\n';j++)
            {
                write(filed,&strip[i][j],1);
                k++;
                if(k>=length)
                {
                  next =1;
                  break;
                }
            }
        }
    }
    while(1)
    {
        if(k<length)
        {
            read(sockfd,&byte,1);
            write(filed,&byte,1);
            k++;
        }
        else
        {
            i=0;
            do{
                read(sockfd,&byte,1);
```

```
                strcat(tamanho,&byte);
            }while(byte!='\n');
            length=strtol(tamanho,NULL,16);
            k = 0;
        }
        if(length==0)
            return;
    }
}

int getFicheiro(int sockfd, char * ficheiro, char*url)
{
    int bytes;
    int fd;
    char answer[1024];
    char ant1,act1,ant2,act2;
    int i,j;


    int length;
    int aux;
    char ** partes;
    int nrpartes;

    //Separar os Headers do resto da informacao
    char ** headers;
    int nrheaders;

    int incache = 0;
    char * lastmodaux;

    int erro;
    int not_modified=0;

    int chunked = check_chunked(partes,nrpartes);
    fd = open(ficheiro, O_CREAT|O_RDWR, S_IRUSR|S_IWUSR);
    bytes = read(sockfd,answer,MAXLEN);


    nrpartes = makeargv(answer,"\r\n\r\n",&partes);

    //Verifica a existencia de erros na resposta do servidor
    i = check_errors(partes[0],nrpartes);

    length = check_length(partes,nrpartes);
    switch(i)
    {
        case 2:
            printf("Estado: HTTP 200 OK\n");
            break;
        case 3:
            printf("Estado: 3xx Redirection\n");
            break;
        case 31:
            printf("Estado: 301 Moved Permanently\n");
            break;
        case 34:
            printf("Estado: 304 Not Modified\n");
            not_modified=1;
            break;
        case 4:
```

```
            printf("Estado: 4xx Client Error\n");
            break;
        case 40:
            printf("Estado: 400 Bad Request\n");
            break;
        case 44:
            printf("Estado: 404 Not Found\n");
            break;
        case 5:
            printf("Estado: 5xx Server Error\n");
            break;
    }
    if(!not_modified){

    //Verifica a data de modificacao do ficheiro e guarda na cache
    lastmodaux= check_modified(partes,nrpartes);
    for(i=0;i<cached;i++)
        if(strcmp(cache[i].path,url)==0)
        {
            incache=1;
            cache[i].lastmodified = lastmodaux;
            break;
        }
    if(!incache)
    {
        cache[cached].lastmodified = lastmodaux;
        cached++;
    }


    for (i = 3; i<=1024; i++)
    {
        ant1 = answer[i-3];
        act1 = answer[i-2];
        ant2 = answer[i-1];
        act2 = answer[i];

        if((ant1==ant2) && (act1==act2) && (act1=='\n') &&
(ant1=='\r'))
        {
            if(chunked)
                get_chunked(fd,answer+i-1,sockfd);
            else
            {
                for(j=0;j<i-2;j++)
                  printf("%c",answer[j]);
                if(length<1024 && length>0)
                {
                  write(fd,answer+i+1,length);
                  aux=length;
                }
                else
                {
                  aux = bytes-i-1;
                  write(fd,answer+i+1,aux);
                }
            }
            break;
        }
    }
    if(!chunked){
```

```
        if(length!=0)
        {
            i = length-aux;
            do
            {
                if(i<=0)
                  break;
                if(i>1024)
                {
                  read(sockfd, answer,1024);
                  write(fd,answer,1024);
                  i-=1024;
                }
                else
                {
                  read(sockfd,answer,i);
                  write(fd,answer,i);
                  break;
                }
            }while(1);
        }
        else
        {
            do
            {
                bytes = read(sockfd,answer,1024);
                write(fd,answer,bytes);
            }while(bytes>0);
        }
    }
    printf("Transferencia concluida\n");
    }
    return 1;
}


int ver_site(char* url, int condicional)
{

    int sockfd;

    //String a ser enviada para o servidor
    char buff[1024];

    //Variáveis para utilização do makeargv
    char ** tokens;
    int ntokens;
    int i;

    //Tratamento do URL
    char filename[50];
    char * directorio;

    char lastmodified[50];
    char * lastmodaux;
    int incache =0;
    //Alterar posteriormente para mudar a porta... DEFAULT: 80
    ntokens = makeargv(url,"/",&tokens);

    //Guardar o path no ficheiro de cache
    for(i=0;i<cached;i++)
```

```
    {
        if(strcmp(cache[i].path,url)==0)
        {
            incache=1;
            break;
        }
    }
    if(!incache)
        cache[cached].path = url;

    //Fazer parse do URL
    if(ntokens>1)
    {
        strtok(url,"/");
        directorio = strtok(NULL,"\0");
    }
    else directorio = " ";

    for(i=0;i<(ntokens-1);i++)
    {
        mkdir(tokens[i],S_IRWXU |S_IRWXG | S_IRWXO);
        chdir(tokens[i]);
    }

    if(ntokens==1)
    {
        mkdir(tokens[0],S_IRWXU|S_IRWXG|S_IRWXO);
        chdir(tokens[0]);
        strcpy(filename,"index.html");
    }
    else
        strcpy(filename,tokens[ntokens-1]);


    printf("Full path: %s/%s\n",url,filename);
    if(condicional)
    {
        if(incache)
        {
            for(i=0;i<cached;i++)
            {
                if(strcmp(cache[i].path,url)==0)
                {
                  lastmodaux = cache[i].lastmodified;
                  incache = 2;
                  break;
                }
            }
            if(incache!=2||(strcmp(lastmodaux,"No Info")==0))
                lastmodaux = "";
        }
        else
            lastmodaux = "";

        i = sprintf(buff,"GET /%s HTTP/1.1\r\nHOST: %s\r\nIf-Modified-
Since: %s\r\n\r\n",directorio,tokens[0],lastmodaux);
        buff[i] ='\0';
    }
    else
    {//Preencher a string a enviar
```

```
        i = sprintf(buff,"GET /%s HTTP/1.1\r\nHOST:
%s\r\n\r\n",directorio,tokens[0]);
        buff[i] ='\0';
    }

    if((sockfd = create_socket(tokens[0],porta))<0)
    {
        perror("create_socket()");
        exit(0);
    }
    freemakeargv(tokens);
    if(write_socket(buff,sockfd)>0)
    {
        printf("Escrito com sucesso\n%s\n",buff);
    }
    if(getFicheiro(sockfd,filename,url)<0)
    {
        perror("getFicheiro()");
    }
    for(i = 0; i<(ntokens-1);i++)
        chdir("..");
    if(ntokens==1)
        chdir("..");

    close(sockfd);
    return 1;
}

int main(int argc, char*argv[])
{
    int sockfd;
    int portaAUX;

    //Variaveis auxiliares
    int i, opcao;

    //bytes lidos
    int bytes;

    //Teste das Sockets
    char pedido[100];

    char answer[MAXLEN];

    FILE * cachefile;


    if((cachefile = fopen("cache","a+"))==NULL)
    {
        perror("fopen()");
        return -1;
    }
    else
        if(fill_cache(cachefile)==-1)
        {
            printf("Erro ao ler o ficheiro de cache.\n");
            return -1;
        }
    fclose(cachefile);
    chdir("cachedir");
    printf("Cliente de HTTP/1.1 iniciado com sucesso!\n");
```

```
    while((opcao=menu())!=6)
    {
        switch(opcao){
            case 1:
                /*Caso simples */
                printf("Por favor escreva o URL do site a visitar:\n");
                fgets(pedido,sizeof(pedido),stdin);
                i=0;
                while(pedido[i]!='\n')
                   i++;
                pedido[i] ='\0';
                ver_site(pedido,0);
                break;
            case 2:
                /* Caso Get -Cond */
                printf("Por favor escreva o URL do site a visitar:\n");
                fgets(pedido,sizeof(pedido),stdin);
                i=0;
                while(pedido[i]!='\n')
                   i++;
                pedido[i]='\0';
                ver_site(pedido,1);
                break;
            case 3:
                /* Sacar a cena toda */
                break;
            case 4:
                /* Utilizando um proxy server, mas isto sao contas de
outro rosario */
                break;
            case 5:
                /* alterar a porta a utilizar */
                printf("Neste momento estamos a utilizar a porta %d\n",
porta);
                printf("Que porta pretende utilizar? Por favor indique
de seguida [Porta deve ser > 1024]:\n");
                fgets(pedido,sizeof(pedido),stdin);
                i=0;
                while(pedido[i]!='\n')
                   i++;
                pedido[i]='\0';
                portaAUX = atoi(pedido);
                if(portaAUX>1024)
                {
                   porta = portaAUX;
                   printf("Nova porta a utilizar é: %d\n",porta);
                }
                else
                   printf("Valor inválido: %d\n Porta definida: %d\n",
portaAUX, porta);
                break;
            default:
                printf("Opcao invalida!\n");
                break;
        }
    }
    chdir("..");
    cachefile=fopen("cache","w");
    save_cache(cachefile);
    fclose(cachefile);
```

```
    return 1;
}
```

servidor.c

```
//FICHEIROS
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <signal.h>

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <netdb.h>
#include <strings.h>
#include <sys/wait.h>
#include <sys/stat.h>

#include <string.h>
#include "../cliente/helper.h"


#include <time.h>

#define MAXLEN 1024
#define SERVER_PORT 8080



#define CONTENT "Content-Type: text/html; charset=iso8859-1"
#define SERVER "Server: Claudio && Simao && Goncalo - HTTP Server -
Beta Version - Catchim"


#define BADREQUEST "HTTP/1.1 400 Bad Request"

#define FILENOTFOUND "HTTP/1.1 404 File Not Found"

#define FILEOK "HTTP/1.1 200 OK"

void fireman(void)
{
     while(waitpid(-1, NULL, WNOHANG)>0)
         ;
}

int getFileSize(int fd)
{
     struct stat file;
     if(!fstat(fd,&file))
     {
         return file.st_size;
     }
     return 0;
}
```

```
int check_host(char**headers,int numero)
{
     int i =0;
     for(i;i<numero;i++)
     {

         if((strcmp(strtok(headers[i],":"),"HOST")==0)&&
(strcmp(strtok(headers[i],":"),"Host")==0))
             return 1;
     }
     return 0;
}

time_t check_modifiedsince(char**headers,int numero)
{
     int i =0;

     time_t req_date;
     struct tm request_date;
     char *dateString;
     for(i;i<numero;i++)
     {

         if((strncmp(strtok(headers[i],": "),"If-Modified-
Since",strlen("If-Modified-Since"))==0))
         {
             dateString = strtok(NULL,"\n");
             printf("A date String: %s\n",dateString);
             if(strptime(dateString," %a %b %d %T
%Y",&request_date)==NULL)
             {
                 return -1;
             }
             return (req_date = mktime(&request_date));
         }
     }
     return -1;
}

int handle_request(int sockfd,char*request)
{

         int filed;

         //Utilizado para "obter" a palavra "GET" do request
         char * token;

         //Variaveis utilizados para fazer o parse do request por
"\r\n"
         char ** stripped;
         int ntokens;

         //Auxilio
         char ** stripped2;
         int ntokens3;

         //Guarda o directorio do ficheiro a ser buscado
         char * directorio;

         //HOST
```

```
        char * pagina;

        //Variaveis utilizadas para fazer o parse do "path"
        char ** path;
        int ntokens2;

        //Guarda a informacao a ser enviada para os clientes
        char answer[1024];

        //Nome do ficheiro .html
        char ficheiro[50];

        int i;

        struct stat file_info;

        char *auxiliar;

        //Variaveis para controlar o Header If-Modified-Since
        time_t oldTime;
        time_t timemodified;
        double diffdate;


        //HTTP/1.1 ?
        char * http;

        time_t now;
        struct tm *tnow;
        time(&now);
        tnow=localtime(&now);



        if((ntokens = makeargv(request,"\r\n",&stripped))==-1)
            return -1;

        chdir("public_html");

        token = strtok(stripped[0], " "); //vai buscar a primeira
palavra, a ver se é um get
        directorio = strtok(NULL," "); //guardar o directorio
        http = strtok(NULL," ");

        //Bad Command
        if(strcmp(token,"GET")!=0)
        {
            sprintf(answer,"HTTP/1.1 4xx Client
Error\r\n%s\r\n%s\r\nDate:%s\r\n",CONTENT,SERVER,asctime(tnow));
            write(sockfd,answer,strlen(answer));
            freemakeargv(stripped);
            freemakeargv(path);
            return -1;
        }
        if(strcmp(http,"HTTP/1.1")!=0)
        {
            sprintf(answer,"400 Error - Bad Request - HTTP Version
1.1!!\n");
            write(sockfd, answer, strlen(answer));
            freemakeargv(stripped);
            freemakeargv(path);
```

```
            return -1;
        }


        //Bad Request 400

        if(check_host(stripped,ntokens))
        {

            filed = open("erros/400",O_RDONLY);
            i=getFileSize(filed);
            sprintf(answer,"%s\r\n%s\r\n%s\r\nContent-Length:
%d\r\nConnection:
close\r\nDate:%s\r\n",BADREQUEST,CONTENT,SERVER,i,asctime(tnow));
            printf("Enviado:\n%s\n", answer);
            write(sockfd,answer,strlen(answer));
            printf("O tamanho total é: %d\n",i);
            do{
                if(i<=0)
                {
                  break;
                }
                if(i>1024)
                {
                  read(filed,answer,1024);
                  write(sockfd,answer,1024);
                  i -=1024;
                }
                else
                {
                  read(filed,answer,i);
                  write(sockfd,answer,i);
                  break;
                }
            }
            while(1);
            close(filed);
            freemakeargv(stripped);
            freemakeargv(path);
            return 1;
        }


        if((ntokens2 = makeargv(directorio,"/",&path))==-1) //Separar
as várias pastas do path
            return -1;

        switch(ntokens2)
        {
            case 0:
                strcpy(ficheiro,"index.html");
                break;
            case 1:
                strcpy(ficheiro,path[0]);
                break;
            default:
                for(i=0;i<(ntokens-1);i++)
                {
                  if(chdir(path[i])==-1)
                  {
```

```
                        filed = open("erros/404",O_RDONLY);
                        i=getFileSize(filed);
                        sprintf(answer,"%s\r\n%s\r\n%s\r\nContent-
Length: %d\r\nConnection:
close\r\nDate:%s\r\n",FILENOTFOUND,CONTENT,SERVER,i,asctime(tnow));
                        printf("Enviado:\n%s\n",answer);
                        write(sockfd,answer,strlen(answer));
                        do{
                              if(i<=0)
                              {
                                    break;
                              }
                              if(i>1024)
                              {
                                    read(filed,answer,1024);
                                    write(sockfd,answer,1024);
                                    i -=1024;
                              }
                              else
                              {
                                    read(filed,answer,i);
                                    write(sockfd,answer,i);
                                    break;
                              }
                        }while(1);
                        close(filed);
                        freemakeargv(stripped);
                        freemakeargv(path);
                        return 1;
                  }
               }
               strcpy(ficheiro,path[i]);
               break;
         }

         ntokens3 = makeargv(request,"\r\n",&stripped2);
         //Verifica a existencia do Header "If-Modified Since"
         timemodified = check_modifiedsince(stripped2,ntokens3);
         if((filed=open(ficheiro,O_RDONLY))!=-1)
         {
             i=getFileSize(filed);
             fstat(filed,&file_info);
             oldTime = (time_t)&file_info.st_mtime;

             if(timemodified>1)
             {
                 diffdate = difftime(oldTime,timemodified);
                 //Not-Modified
                 if(diffdate<0)
                 {
                   printf("Estado: Ficheiro nao modificado\n");
                   sprintf(answer,"HTTP/1.1 304 Not Modified\r\nDate:
%s%s\r\n%s\r\nContent-Length: 0\r\n",asctime(tnow),SERVER,CONTENT);
                   printf("Enviado:\n%s\n",answer);
                   write(sockfd,answer,strlen(answer));
                   freemakeargv(stripped);
                   freemakeargv(stripped2);
                   freemakeargv(path);
                   return 1;
                 }
             }
```

```
            sprintf(answer,"%s\r\n%s\r\n%s\r\nContent-Length:
%d\r\nLast-Modified: %sConnection:
close\r\nDate:%s\r\n",FILEOK,CONTENT,SERVER,i,ctime(&file_info.st_mtim
e),asctime(tnow));
            printf("Enviado:\n%s\n",answer);
            write(sockfd,answer,strlen(answer));
            do{
                if(i<=0)
                {
                  break;
                }
                if(i>1024)
                {
                  read(filed,answer,1024);
                  write(sockfd,answer,1024);
                  i -=1024;
                }
                else
                {
                  read(filed,answer,i);
                  write(sockfd,answer,i);
                  break;
                }
            }while(1);
            close(filed);
            freemakeargv(stripped2);
            freemakeargv(stripped);
            freemakeargv(path);
            return 1;
        }
        else
        {
            filed =open("erros/404",O_RDONLY);
            i=getFileSize(filed);
            sprintf(answer,"%s\r\n%s\r\n%s\r\nContent-Length:
%d\r\nConnection:
close\r\nDate:%s\r\n",FILENOTFOUND,CONTENT,SERVER,i,asctime(tnow));
            printf("Enviado:\n%s\n",answer);
            write(sockfd,answer,strlen(answer));
            do{
                if(i<=0)
                {
                  break;
                }
                if(i>1024)
                {
                  read(filed,answer,1024);
                  write(sockfd,answer,1024);
                  i -=1024;
                }
                else
                {
                  read(filed,answer,i);
                  write(sockfd,answer,i);
                  break;
                }
            }while(1);
            close(filed);
            freemakeargv(stripped2);
            freemakeargv(stripped);
            freemakeargv(path);
```

```
            return 1;
        }

        freemakeargv(stripped2);
        freemakeargv(stripped);
        freemakeargv(path);
}
int main(int argc, char**argv)
{
     int sockfd, newsockfd, client_size;

     struct sockaddr_in server_addr, client_addr;
     int childpid;
     char buf[MAXLEN];
     int bytes;

     signal(SIGCHLD, fireman);

     /*open an TCP socket*/
     if ((sockfd = socket(AF_INET,SOCK_STREAM,0)) < 0) {
        perror("socket()");
        exit(0);
     }
     /*bind the local address so that a client can connect*/
     bzero((char*)&server_addr,sizeof(server_addr));
     server_addr.sin_family = AF_INET;
     server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
     server_addr.sin_port = htons(SERVER_PORT); /*server TCP port must
be network byte ordered */

     if (bind(sockfd,
          (struct sockaddr *)&server_addr,
          sizeof(server_addr)) < 0){
        perror("bind()");
        exit(0);
     }

     printf("\n  -----------------------------------------------\n ");
     printf(" -- RCOM | Redes de Computadores // FEUP 2007 --\n ");
     printf(" --                                            --\n ");
     printf(" -- Servidor HTTP | ei01091.ei03032.ei04100   --\n ");
     printf(" -----------------------------------------------\n ");
     printf("\n");
     listen(sockfd, 5);


     for(;;) {
        /*wait for a connection request (concurrent server)*/
        client_size = sizeof(client_addr);
        /*blocks until a connection request is done*/
        /*then returns the new socket descriptor, now complete*/
        newsockfd = accept(sockfd,
                        (struct sockaddr *)&client_addr,
                   &client_size);
        if (newsockfd < 0) {
            perror("accept()");
            exit(0);
        }
        /*fork the server so one process handle the connection
        and other keeps waiting incoming connection requests*/
        if ((childpid = fork()) < 0) {
```

```
            perror("fork()");
            exit(0);
        }
        else if (childpid == 0){   /*child process*/
            close(sockfd);    /*continue processing with the new
descriptor*/
            bytes = read(newsockfd, buf, MAXLEN);
            printf("Bytes lidos = %d\n%s\n", bytes, buf);
            handle_request(newsockfd,buf);
            exit(0);
        }
    }
    return 1;
}
```

makefile cliente

```
BIN=clienteHTTP
OBJ=cliente.o helper.o socket.o
CC = gcc
#CCFLAGS
all: $(BIN)

$(BIN): $(OBJ)
    $(CC) -o $(BIN) $(OBJ)

clean:
    rm *.o
    rm cache
    rm -r cachedir/*
```

makefile servidor

```
BIN=servidorHTTP
OBJ=servidor.o ../cliente/helper.o
CC = gcc
#CCFLAGS
all: $(BIN)

$(BIN): $(OBJ)
    $(CC) -o $(BIN) $(OBJ)

clean:
    rm *.o
```