Architecture and Network Programming

8sif120

Lab #5

(due date ==>Oct 24th 2020)

Paul Girard, Ph.D.

Part 1: Learn http with a telnet client and a commercial http server
Part 2: Design a basic http server with a commercial browser
(Firefox, Chrome, Opera, Edge) strongly recommended, not IE)

Objective of Part 1 *(no lab report for this part)*
Learn the basics of HTTP/1.0 & 1.1 protocol by using the GET method with some client header parameters to handle a client telnet session with a commercial web server *(ex. Apache)* for downloading an html file.

Methodology

1. Use a client web browser to visualize the file *page1.html* and the source code: http://dim-ensxcn1.uqac.ca/~pgirard/page1.html or

2. Leave the browser on this page and run Wireshark. **Start** the capture and click on **Refresh** from the browser. Verify the ethernet frame having the HTTP protocol with the command GET as the data part. Verify the format and the content of this client request. These header parameters are sent to the HTTP server. Your understanding of these lines will facilitate your task in writing your own HTTP server. Display the TCP stream.

3. Start a **telnet** client with putty on dim-ensxcn1 and create a connection with a commercial HTTP server on dim-ensxcn2 listening on port TCP 80 (*telnet dim-ensxcn1.uqac.ca 80).* Telnet is like SSH but it is unsecure. As long as you are speaking the HTTP language, the server will respond correctly.

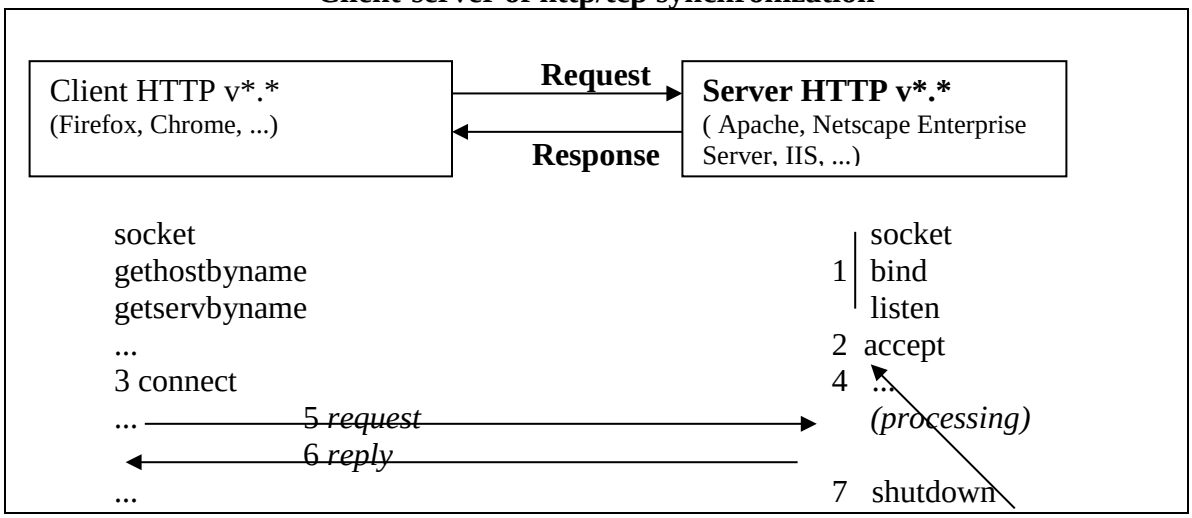
4. Enter the necessary command and headers to get the page *page1.html* with the command **GET**. Identify yourself in the header **From** with your email address and specify a name and a version to your client program in the header **User-Agent.** A carriage return *(CR)* on an empty line terminates the request.

GET /~pgirard/page1.html HTTP/1.0 ***(CR)***	or **GET /~pgirard/page1.html HTTP/1.1** **Host: dim-ensxcn1.uqac.ca** *User-Agent:* ****** *From:* ******** ***(CR)***

5. Now try to identify each line:
   - the information created by the telnet protocol,
   - the client command sent by the HTTP client,
   - the headers and body of the HTTP server's response,
   - the name and version of the HTTP server's and
   - the format of the received document *page1.html*

Part 1
Appendix A

Client-server of http/tcp synchronization



Sequence of steps

- The server initializes and waits for a client connection by listening tcp port 80
- The client asks for a ressource URL
  (ex http://dim-ensxcn1.uqac.ca/~pgirard/page1.html)
- The client browser asks to a DNS server the IP address of the requested web server
- The DNS server responds with the requested server's address *(ex. 132.212.240.20)*
- The client browser tries to establish a tcp connection on port 80 on the server (connect)
- If the connection is accepted, the client browser sends its request *(command, headers)* to the web server *(ex. GET /~pgirard/page1.html,...)*
- The HTTP server validates the request, reads the requested information, sets the reply header and sends the reply with the html file if there is no error; in case of error, an http error is sent *(ex. ERROR 404).*
- The TCP connection is closed by the web server at the end of transmission unless the client header specified to leave open the connection *(Connection: Keep-Alive).* But the web server has always the choice to close it and limits each time the delay.
- The browser displays the text *page1.html.*
- The browser sends a new request to get *image.jpg* with *GET .../image.jpg* ; once received, it displays the image after the decoding of the jpeg algorithm.
- The connection is then closed by the web server.

Part 1

Appendix B

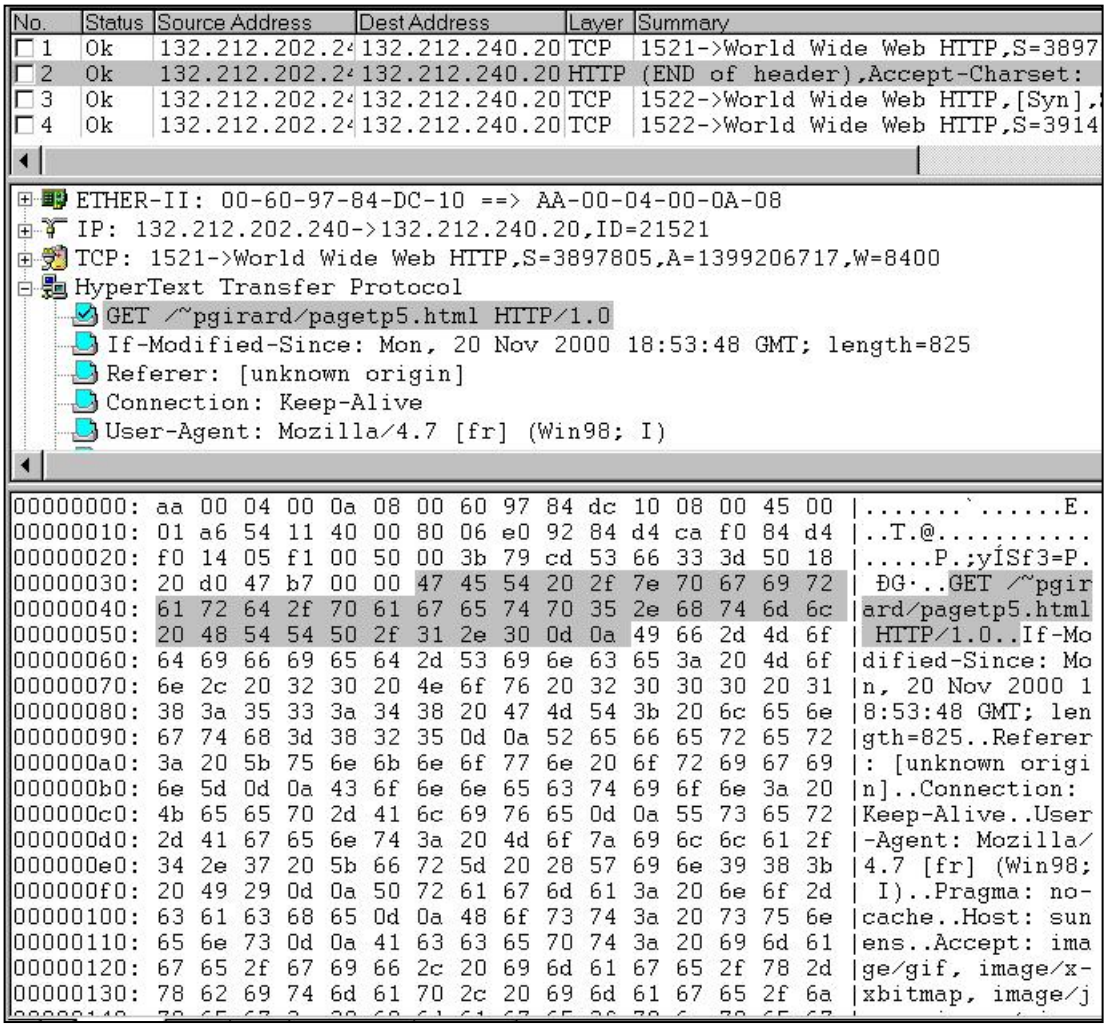
page1.html displayed

Source of document

```
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<meta name="Author" content="Paul Girard"><meta name="GENERATOR"
content="Mozilla/4.7 [fr] (Win98; I) [Netscape]"><title>page1</title>
</head><TITLE>8sif115 - Lab 5</TITLE>
<body text="#000000" bgcolor="#FFFFC0" link="#0000FF" vlink="#800080"
alink="#FF00FF"><blockquote><blockquote>
<center><b><font color="#FF6666"><font size=+3>8sif115</font></font></b>
<br><b><font color="#FF6666"><font size=+3>Lab 5</font></font></b><b><font
color="#FF6666"><font size=+3></font></font></b>
<p><b><font color="#FF6666"><font size=+3>Very simple html page
</font></font></b>
<p><img SRC="lake.jpg" height=200 width=300>
<p><b><i>(Paul Girard)</i></b></center></blockquote></blockquote>
</body></html>
```

Part 1
Appendix C

Packet containing the request
http://dim-ensxcn1.uqac.ca/~pgirard/page1.html



No.	Status	Source Address	Dest Address	Layer	Summary
1	Ok	132.212.202.24	132.212.240.20	TCP	1521->World Wide Web HTTP,S=3897
2	Ok	132.212.202.24	132.212.240.20	HTTP	(END of header),Accept-Charset:
3	Ok	132.212.202.24	132.212.240.20	TCP	1522->World Wide Web HTTP,[Syn],
4	Ok	132.212.202.24	132.212.240.20	TCP	1522->World Wide Web HTTP,S=3914

```
ETHER-II: 00-60-97-84-DC-10 ==> AA-00-04-00-0A-08
IP: 132.212.202.240->132.212.240.20,ID=21521
TCP: 1521->World Wide Web HTTP,S=3897805,A=1399206717,W=8400
HyperText Transfer Protocol
  GET /~pgirard/pagetp5.html HTTP/1.0
  If-Modified-Since: Mon, 20 Nov 2000 18:53:48 GMT; length=825
  Referer: [unknown origin]
  Connection: Keep-Alive
  User-Agent: Mozilla/4.7 [fr] (Win98; I)
```

```
00000000: aa 00 04 00 0a 08 00 60 97 84 dc 10 08 00 45 00 |.......`......E.
00000010: 01 a6 54 11 40 00 80 06 e0 92 84 d4 ca f0 84 d4 |..T.@...........
00000020: f0 14 05 f1 00 50 00 3b 79 cd 53 66 33 3d 50 18 |.....P.;yÍSf3=P.
00000030: 20 d0 47 b7 00 00 47 45 54 20 2f 7e 70 67 69 72 | ÐG·..GET /~pgir
00000040: 61 72 64 2f 70 61 67 65 74 70 35 2e 68 74 6d 6c |ard/pagetp5.html
00000050: 20 48 54 54 50 2f 31 2e 30 0d 0a 49 66 2d 4d 6f | HTTP/1.0..If-Mo
00000060: 64 69 66 69 65 64 2d 53 69 6e 63 65 3a 20 4d 6f |dified-Since: Mo
00000070: 6e 2c 20 32 30 20 4e 6f 76 20 32 30 30 30 20 31 |n, 20 Nov 2000 1
00000080: 38 3a 35 33 3a 34 38 20 47 4d 54 3b 20 6c 65 6e |8:53:48 GMT; len
00000090: 67 74 68 3d 38 32 35 0d 0a 52 65 66 65 72 65 72 |gth=825..Referer
000000a0: 3a 20 5b 75 6e 6b 6e 6f 77 6e 20 6f 72 69 67 69 |: [unknown origi
000000b0: 6e 5d 0d 0a 43 6f 6e 6e 65 63 74 69 6f 6e 3a 20 |n]..Connection:
000000c0: 4b 65 65 70 2d 41 6c 69 76 65 0d 0a 55 73 65 72 |Keep-Alive..User
000000d0: 2d 41 67 65 6e 74 3a 20 4d 6f 7a 69 6c 6c 61 2f |-Agent: Mozilla/
000000e0: 34 2e 37 20 5b 66 72 5d 20 28 57 69 6e 39 38 3b |4.7 [fr] (Win98;
000000f0: 20 49 29 0d 0a 50 72 61 67 6d 61 3a 20 6e 6f 2d | I)..Pragma: no-
00000100: 63 61 63 68 65 0d 0a 48 6f 73 74 3a 20 73 75 6e |cache..Host: sun
00000110: 65 6e 73 0d 0a 41 63 63 65 70 74 3a 20 69 6d 61 |ens..Accept: ima
00000120: 67 65 2f 67 69 66 2c 20 69 6d 61 67 65 2f 78 2d |ge/gif, image/x-
00000130: 78 62 69 74 6d 61 70 2c 20 69 6d 61 67 65 2f 6a |xbitmap, image/j
```

Request using a telnet client on port 80 *(listened by Apache http server)*
(check the client headers and the commmand)

```
dim-ensxcn1:pgirard> telnet dim-ensxcn1.uqac.ca 80
Trying 132.212.240.20...
Connected to dim-ensxcn2.
Escape character is '^]'.

GET  /~pgirard/page1.html  HTTP/1.0
 (empty line)
```

Reply from the server
(server headers &, body)

```
HTTP/1.1 200 OK
Date: Wed, 06 Nov 2002 21:17:03 GMT
Server: Apache/1.3.19 (Unix) PHP/4.0.6
Last-Modified: Mon, 20 Nov 2000 18:53:48 GMT
ETag: "44bcd-339-3a19733c"
Accept-Ranges: bytes
Content-Length: 825
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug
```

```
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html><head><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<meta name="Author" content="Paul Girard">
<meta name="GENERATOR" content="Mozilla/4.7 [fr] (Win98; I) [Netscape]">
<title>page1</title></head>
<body text="#000000" bgcolor="#FFFFC0" link="#0000FF" vlink="#800080" alink="#FF
00FF"><blockquote><blockquote><center><b><font color="#FF6666"><font size=+3>
8inf115</font></font></b><br><b><font color="#FF6666"><font size=+3> Lab 5 </font>
</font></b><b><font color="#FF6666"><font size=+3></font></font></b><p> <b>
<font color="#FF6666"><font size=+3>Very simple html page</font></font></b><p>
<img SRC="lake.jpg" height=200 width=300>
<p><b><i>(Paul Girard)</i></b></center></blockquote></blockquote></body></html>
Connection closed by foreign host.

dim-ensxcn1:pgirard>
```

Part 2

(lab report and programming)

Objective of part 2

Starting from the downloadable source program *server_web120.c* already supporting some basic web server functions, modify this program to support more advanced web functions that has to be supported by a commercial browser.

Specifications of the server_web.c program

1. Rename this program **server_web.c** with the command **mv***(the executable will be server_web)*. Delete the code *#define Home_Page* in the source program and replace it by a new file *index.html* located in the same directory as the server *(main directory)*. Use the same html code without the special C delimiter *(ex. \/ becomes /)*. Leave the code defining the html page */time* in your program. This code could be used for a time server.

2. Login to **dim-ensxcn1.uqac.ca** with your user account.
   - copy with the **cp** command all needed files to do lab5 from my user account:

   cp ../web00300/* .

   if you do the command **ls**, you should see 6 new files

```
[web00***@dim-ensxcn1 ~]$ ls
server_web120.c, compweb120, page1.html, image.gif, image.jpg, favicon.ico
```

   You should download these files with EditPlus to keep a backup on a USB key.

   Use the command file *compweb120* to compile your basic web program. Don't forget to change the privilege of *compweb120* with *chmod 700* compweb120 to make it executable.

   note : All new browsers *(Firefox,Chrome,...)* will sometimes ask for **favicon.ico** which is an icon *(graphics usually 18x18 pixels)* that MUST BE on the same directory as your web server. You have no problem with a commercial web server but with Lab5 you should have one to prevent problems.

3. Create a directory ***document*** *(mkdir document)* from your main directory and copy in this directory once again the 3 files *page1.html, image.gif, image.jpg.* With EditPlus, personalize *page1.html* by inserting your name in the page.

```
> mkdir document
> cd document
> cp ../image.* .
> cp ../page1.html .
> cd ..
```

New functions to be added to server_web

4. If the requested file is not mentioned *(path = "/")*, use the default file *index.html* *(ex. http://dim-ensxcn1.uqac.ca:5001/* or *GET / HTTP/1.0).*
   If *path* is not specified, send the message "error 404" to the client and close the client connection.

5. If the file extension is not entered, use the default extension .**html**.
   ex. http://dim-ensxcn1.uqac.cas:5001/page1 means
   http://dim-ensxcn1.uqac.ca:5001/page1.html

6. If the path contains a backward reference, ***("..")*** *(not secure on a server),* send an *error message 400* and close the client connection.
   (ex. http:// dim-ensxcn1.uqac.ca:5001/../document/page1.html) is illegal

7. As soon as the server's response is completed *(code 200, 400 or 404),* close the client connection and wait for another connection on the known port. The server MUST NEVER STOP.

8. The function *send-header* from the server contains the parameter *Content-Type* specifying the type of the file requested, which is determined by the file extension. Three file types will be supported with their own extensions : *html, jpeg & gif.* The parsing of the string *path* will identify the extension. The value taken by *Content-Type* is specified in the following table according to the HTTP protocol. This header parameter will later tell the client how to decode the file.

File Extension	Parameter *Content-Type*
- *html, HTML, htm, HTM*	*Content-Type: text/html*
- *JPEG, jpeg, JPG* or *jpg*	*Content-Type: image/jpeg*
- *gif* or *GIF*	*Content-Type: image/gif*

9. The server header parameter *Content-Length* specifies to the client the size of the requested file in octets *(gif, jpeg ou html).* You'll need to use the *stat()* function and access the field *st_size* in the structure *stat* explained in Appendix D. The client browser will then be able to know the memory size needed by the transfer.

10. Don't forget that an html file is read like a text file using the function *fgets* but a *jpeg* or *gif* file is read in a binary mode with the *fread* function explained in Appendix D.

11. For each client connection, the server displays a message specifying the beginning of a client connection, the request type, the file path and the version of the HTTP protocol used after the analysis of the request.

12. The versions of the protocol HTTP/1.0 ou HTTP/1.1 will all be accepted in the client request GET but all other client header parameters will be ignored *(server_web120 does that).* One can then assume the absence of client restriction.

Note: The following C functions could be necessary *(+ fread(), stat() & feof())*

int strcmp(char *str1, char *str2) compares 2 null terminated strings character by character *(if equal, returns 0)*

int strncmp(char *str1, char *str2, size_t n) compares n first characters taken from 2 null terminated strings *(if equal, returns 0)*

char *strcpy(char *destination, char *source) copies the contents of the string *source* to the string *destination*

char *strstr(char *str1, *str2) locates the first occurrence of the string *str2* in the string *str1* ; it returns a pointer to the beginning of the first occurrence or null if not found.

char *strrchr(char *str, int ch) locates the last occurrence of the character *ch* in the string *str*

char *strcat(char *str1, char *str2) appends the contents of the string *str2* to the end of the string *str1*.

size-t strlen(char *str) returns the number of characters in *str* without the null character.

LAB REPORT

1. On the **first page** and using Word (not a pdf), print 1) each name and TUT student code in your group, 2) one email address, and 3) the user account and password used on dim-ensxcn1.uqac.ca *(web00***).* Insert a **listing** of *server_web.c* and a **sequential** execution for each of the six following URL *(use a complete screen capture for each one).*

   - http:// dim-ensxcn1.uqac.ca:no_port/
   - http:// dim-ensxcn1.uqac.ca:no_port/time
   - http:// dim-ensxcn1.uqac.ca:no_port/page1
   - http:// dim-ensxcn1.uqac.ca:no_port/document/image.jpg
   - http:// dim-ensxcn1.uqac.ca:no_port/document/image.gif

2. Send this report by email to pgirard@uqac.ca. Penalty will begin after this date. Make sure to receive an acknowledge from me after your email to confirm **but wait at least 4 days before retransmitting**. **Do not send your report many times everyday**. If I have some problems I will send an message on the web site *(important note)*

   http://www.uqac.ca/pgirard/english_version/sif120/remark.html

Results will be displayed on the web server as soon as possible with lab1, lab5 and all exams . It takes at least 3 weeks to evaluate all labs.

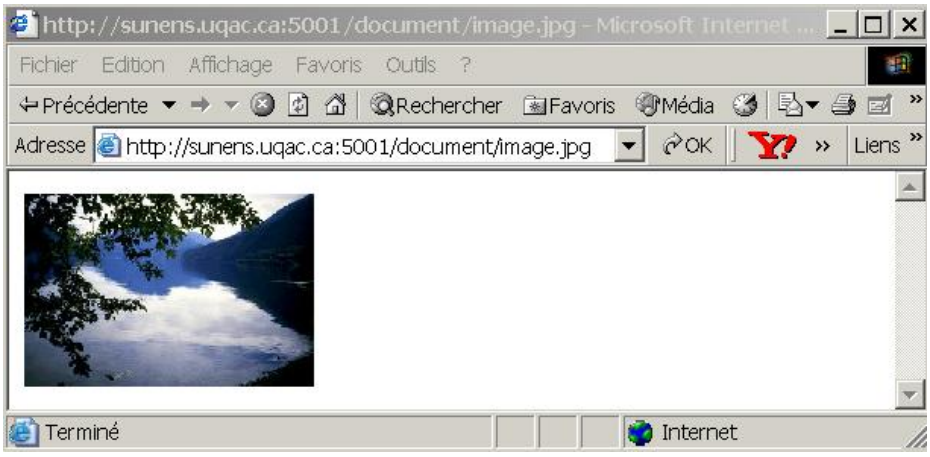
IMPORTANT

➔ **Find the extension of a file, DO NOT use a specific file name in your source C program (there will be a penalty)**

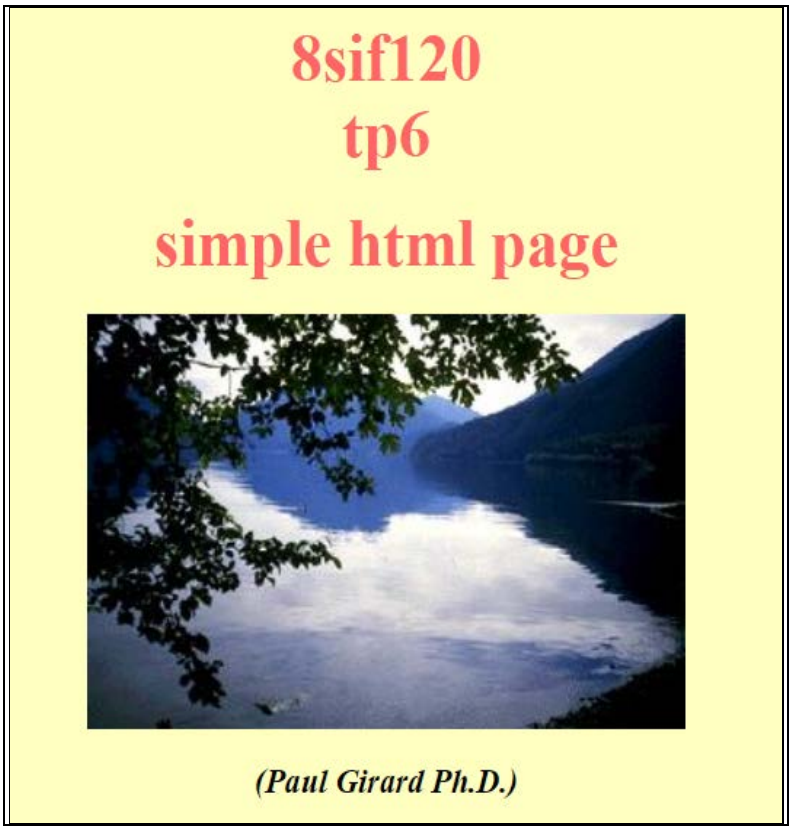
➔ **If you copy another student's work, you will have 0**

➔ **DO NOT REMOVE ANY ACTUAL COMMENTS BUT YOU MAY ADD YOUR OWN COMMENTS. No evaluation if you do that.**

Part 2 : Appendix C
Examples




http://sunens.uqac.ca:5001/index.html - Microsoft Internet Explorer
Fichier Edition Affichage Favoris Outils ?
Précédente
Rechercher
Favoris
Média
Adresse
http://sunens.uqac.ca:5001/index.html
OK
Liens
Welcome to mini-server PG_web
You may visit :
my personal web site
Internet


8sif120
tp6
simple html page
(Paul Girard Ph.D.)

Example of a server_web log

```
dim-ensxcn1:pgirard>server_web 5001
The TCP socket port is  #5001
Server ready for connections

Client connection
Request type: GET          path : /document/image.jpg          version : HTTP/1.1
Number of octets sent: 4353,
File read
End of client connection

Client connection
Request type : GET          path : /index                       version : HTTP/1.1
filename : index1.html
Error opening file index1.html
End of client connection
```

Example of a client connection using a telnet session

```
dim-ensxcn2:pgirard> telnet dim-ensxcn2.uqac.ca 5001
Trying 132.212.240.20...
Connected to sunens.
Escape character is '^]'.
GET /time HTTP/1.0

HTTP/1.0 200 OK
Server: Server HTTP V1.0 (Paul Girard, UQAC)
Content-Length: 105
Content-Type: text/html

<head></head><body><html><h1>The actual date/time is : <br><br>Wed Nov 20
13:31:092
</h1></html></body>
Connection closed by foreign host.

dim-ensxcn2:pgirard>
```

Example of a client request connection using a telnet session

```
dim-ensxcn2:pgirard> telnet dim-ensxcn2.uqac.ca 5001
Trying 132.212.240.20...
Connected to sunens.
Escape character is '^]'.
GET / HTTP/1.1

HTTP/1.0 200 OK
Server: Mini server web V1.0, Paul Girard (UQAC)
Content-Length: 219
Content-Type: text/html

<head></head><body><html><h1>Welcome to mini-server PG_web</h1><p>You may visit
: <ul><li><a href="http://www.uqac.ca/~pgirard"><font color="#0000
ff"> my personal web site </font></a></li></ul></html></body>
Connection closed by foreign host.
```

Part 2
Appendix D
Description of C functions

fread(), feof()

Format

#include <stdio.h>

size_t fread(void *_buf_, size_t _size_, size_t _nitems_, FILE *_ptr_);

int feof(FILE *_ptr_);

Arguments

buf	is a pointer to an array of *nitems* elements each of which is *size* characters long containing read elements of file pointed by *ptr*
size	is the size of one element *(usually one octet)*
nitems	is the maximum number of elements to can be read
ptr	is a pointer to a binary file

Description

fread() reads up to *nitems* elements of the indicated *size* from the input stream into the specified array. The actual number of octets read is returned by *fread().* It may be less than *nitems* if EOF is encountered on the file. If *nitems* is encountered, *fread()* stops reading and the pointer *ptr* is incremented to the next element in the binary file. **feof(ptr)** returns 0 if EOF has not been detected by *ptr*.

Return value

- Upon successful completion, *fread()* returns the number of elements successfully read.
- If *size* or *nitems* is 0, *fread()* returns 0
- Otherwise, if a read error occurs, the error indicator *errno* is set to indicate the error.

Example

```
#define BUFFSIZE 256
FILE    *ptr;                   /* file descriptor */
char    buf[BUFFSIZE];           /* buffer */
int     rval;                   /* # octets read   */
char    filename[] = "image.jpeg";
. . .
if ((ptr = fopen(filename,"r")) == NULL) exit(0);
...
while(feof(ptr)==0) { rval = fread(buf, sizeof(char), BUFFSIZE, ptr);
        if (rval > 0) process the data; ... }
fclose(ptr);
```

stat()

Format

#include **<sys/types.h>**
#include **<sys/stat.h>**

int stat(const char * *path*, struct stat * *buf*);

Parameters

path points to a string containing the path of the directory or file for which status is to be obtained

buf points to the *stat* structure containing information about the file

Description

stat() gets a *stat* structured information on a file pointed by *path*.

Stat structure

```
mode_t  st_mode;    /* File mode (see mknod(2)) */
ino_t   st_ino;     /* Inode number */
dev_t   st_dev;     /* ID of device containing */
                    /* a directory entry for this file */
dev_t   st_rdev;    /* ID of device */
                     /* This entry is defined only for */
                     /* char special or block special files */
nlink_t st_nlink;   /* Number of links */
uid_t   st_uid;     /* User ID of the file's owner */
gid_t   st_gid;     /* Group ID of the file's group */
off_t   st_size;    /* File size in bytes */
time_t  st_atime;   /* Time of last access */
time_t  st_mtime;   /* Time of last data modification */
time_t  st_ctime;   /* Time of last file status change */
                    /* Times measured in seconds since */
                    /* 00:00:00 UTC, Jan. 1, 1970 */
long    st_blksize; /* Preferred I/O block size */
blkcnt_t st_blocks; /* Number of 512 byte blocks
                       allocated*/
```

Return value

- If successful, stat() returns 0
- If error, stat() returns -1

Example

```
struct   stat buf;
char     filename[] = "image.jpeg";

if(stat(filename, &buf) != 0) exit(1);
printf("File size : %d octets ", buf.st_size);
...
```

Part 2: Appendix E
Suggestion (pseudo-coding)

#include <sys/stat.h>

copy contents of #define HOME_PAGE to file index. html with Edit plus, get rid of C specific characters and build index.html

delete #define HOME_PAGE after copying the contents to a new file named index.html

After validation of GET and HTTP/1.* if OK then continue else error 400;

If path contains ".." then error 400;

If strlen(path) <1 error 400

If strlen(path) = 1 then filename = "index.html" and extension = text/html;

If strlen(path) > 1 then find_name_of_extension; (extension is after the point ".")

If no extension name *(no "." in path)* then extension ="html" ; goto open the file;

if extension = "JPEG or jpeg or JPG or jpg" then content_type = image/jpeg
else if extension ="GIF or gif" then content_type = image/gif
else if extension ="HTML or html or HTM or htm" then content_type=text/html;

openfile: Open filename; if opening error then generate error 404;

find the size of filename with **stat**(filename) then send_header with code 200 OK;

if jpeg or gif then read file in binary and send to socket else read file in ASCII and send to socket until eof

close client connection

note: A new function called **find_extension** would help to get a clean code in the main section.

Part 2: Appendix F
Listing of server_web120.c

```
/*
* Program:      server_web120.c
*  Author:      Paul Girard Ph.D., DIM, UQAC
*  Date:        Sept 14 2020
* Version:      3.6
*
*  Objective:   This basic web program supports http 1.0 et 1.1 :
*          - GET method from a client commercial web navigator
*                                  - validates the request GET for error codes 400 and 404
*                                  - and generates a response following http protocol rules
*                                  (code 200).  html pages are coded in the program source.
*
*                          This version doesn't support access to external files.
*                          Most http client header parameters are ignored :
*                          (User_Agent, From, Referer, Connection, Accept, ...)
*
*       Compilation and link options : (Solaris)
*               gcc server_web120.c -lsocket -lnsl -o server_web120 or
*
*               gcc server_web120.c -lsocket -lnsl -w -o server_web120
*                                                  to suppress "warnings"

        Compilation and link options : (Linux)
*               gcc server_web120.c -lnsl -o server_web120 or
*
*               gcc server_web120.c -lnsl -w -o server_web120
*                                                  to suppress "warnings"
*
*  Execution: 2 formal parameters to run the server :
*                                                   server_name and port #
*                                  dim-ensxcn1% server_web120 port#
*                                  (note: port 80 is assigned to a commercial http server
*               ex.        dim-enszxcn1% server_web120 5001
*
*               The following html pages are coded directly in the source :
*               HOME_PAGE and TIME. They can be called
*               like this by a telnet client or a web client navigator
*
*               1. with a dim-ensxcn1 telnet client using http 1.0 protocol:
*                          telnet dim-ensxn2.uqac.ca  port#   (no_port: port # used)
*                          GET / HTTP/1.0          to get HOME_PAGE
*                          cr/lf                   and an empty line to send the request
*
*                          telnet dim-ensxcn2.uqac.ca  port#
*                          GET /time HTTP/1.0      to get TIME
*                          cr/lf
*
*               2. with telnet using HTTP/1.1 protocol and Apache:
*                          telnet dim-ensxcn2.uqac.ca 80
*                          GET / HTTP/1.1
*                          Host: dim-ensxcn1.uqac.ca
*                          cr/lf
*                          telnet dim-ensxcn1.uqac.ca 80
```

```
*                       Host: dim-ensxcn1.uqac.ca
*                       GET /time HTTP/1.1
*                       Host: dim-ensxcn1.uqac.ca
*                       cr/lf
*
*               3. with Netscape or Explorer client
*                       http://dim-ensxcn1.uqac.ca :no_port/                    to get Home_Page
*                       http://dim-ensxcn1.uqac.ca :no_port/time        to get time
*/

/*
*       Files to include
*/

#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <time.h>

#define BUFFSIZE        256
#define SERVER_NAME             "Mini web server  V1.0 (Paul Girard, Ph.D., UQAC)"

#define ERROR_400       "<head></head><body><html><h1>Error 400</h1><p>Th\
e server couldn't understand your request.</html></body>\n"

#define ERROR_404       "<head></head><body><html><h1>Error 404</h1><p>Do\
cument not found.</html></body>\n"

#define HOME_PAGE       "<head></head><body><html><h1>Welcome to the mini \
web server PG_web</h1><p>Let me invite you to visit : <ul><li><a href=\
\"http:\/\/www.uqac.ca/~pgirard\"><font color=\"#0000ff\"> my personal \
site</font></a></li></ul></html></body>\n"

#define TIME_PAGE       "<head></head><body><html><h1>The actual date-time is : \
<br><br>%s</h1></html></body>\n"

int recvln(int, char *, int);           /* read a client http line  */
void send_header(int, int, int);        /* send a http header to client*/
int wait_connection(short);             /* wait for a client connection */
void send_eof(int);                     /* close a client connection */

int     init = 0;                       /* initialization */
int     sock;                           /* socket */
int     temp;

/* ************************************************************************
*       Main Program
*  ************************************************************************
*/

int main(int argc, char *argv[])    /* argv[0] pointer to program name
                                                        argv[1] pointer to port number */
```

```
{
        int             conn;           /* socket descriptor */
        int             n;
        char    buff[BUFFSIZE], cmd[16], path[64], vers[16];
        char    *timestr;
        struct timeval  tv;             /* delay in sec since 1/1/1970 */
        time_t a;                       /* Linux CentOS function time */
        if (argc != 2)
        {
                fprintf(stderr, "usage : %s <no_port_TCP>\n", argv[0]);
                return 1;
        }

        while(1)
        {

/*              Wait for a client connection on the known port # */

                conn = wait_connection(atoi(argv[1]));
                if (conn < 0) return 1;

/*              Reads & validates the request client */

                n = recvln(conn, buff, BUFFSIZE);
                sscanf(buff, "%s %s %s", cmd, path, vers);
                printf("cmd : %s path : %s vers : %s\n",cmd, path, vers);

/*              Ignore other client header parameters until \r\n on a line */

                while((n = recvln(conn, buff, BUFFSIZE)) > 0)
                {
                        if (n == 2 && buff[0] == '\r' && buff[1] == '\n')
                                break;
                }

/*              Check if there is an unexpected EOF */

                if (n < 1)
                {
                        send_eof(conn);
                        continue;
                }

/*              Validates the http client request  */

                if (strcmp(cmd, "GET") != 0 &&
                        (strcmp(vers, "HTTP/1.0") != 0 ||
                         strcmp(vers, "HTTP/1.1") != 0))
                {
                        send_header(conn, 400, strlen(ERROR_400));
                        send(conn, ERROR_400, strlen(ERROR_400),0);
                        send_eof(conn);
                        continue;
                }

/*              Transmit back the requested html page or
```

```
*                   the message "not found" error */

                    if (strcmp(path, "/") == 0)                 /* default HOME_PAGE*/
                    {
                            send_header(conn, 200, strlen(HOME_PAGE));
                            send(conn, HOME_PAGE, strlen(HOME_PAGE),0);
                    }
                    else if (strcmp(path, "/time") == 0)        /* time page */
                             {
                            /*      Solaris ==>             if (gettimeofday(&tv, NULL) == -1)
                                            puts("Error with gettimeofday()");
                                            timestr = ctime(&tv.tv_sec);        */

                            /* testtime.c: test of time function*/
                                    time(&a);                   /* sec since 1970 */
                                    timestr = ctime(&a);        /* formatted date-time */
                                    sprintf(buff, TIME_PAGE, timestr);
                                    send_header(conn, 200, strlen(buff));
                                    send(conn, buff, strlen(buff), 0);
                                    } else
                        {           /* Error not found */
                                    send_header(conn, 404, strlen(ERROR_404));
                                    send(conn, ERROR_404, strlen(ERROR_404),0);
                        }

                    send_eof(conn);                     /* close the client socket */
            }       /* end of infinite loop */
}                   /* ==== end of main program */

/* ************************************************************************
*       Function wait_connection() returns a client connection
*  ************************************************************************
*/

int wait_connection(short no_port)
{
            struct sockaddr_in sockaddr;        /* internet format */
      int newsock;              /* socket descriptor */
            size_t length;              /* #octets in sockadr_in structure*/

/*
*           Tcp socket creation
*/

            switch (init)
            {
            case 0 :
            sock = socket(AF_INET, SOCK_STREAM, 0);
            if (sock < 0)
                                {
                    perror("Error in socket creation");
                    return 1;
            }

/*
*                   2. Specify the local part of the address :
```

```
*                   1)the local port number in network format and
*                   2)the local IP address.  INADDR_ANY is used on a server
*                   because many ip addresses may be used on the same machine.
*/
        sockaddr.sin_family = AF_INET;
        sockaddr.sin_addr.s_addr = INADDR_ANY;
        sockaddr.sin_port = htons(no_port);

        if (bind(sock, (struct sockaddr *)&sockaddr, sizeof(sockaddr)) < 0)
                        {
                        perror("The bind socket did not work");
                close(sock);
                return 1;
        }

/*
*               Simple validation: find the name associated to this socket
*               and print its port number
*/
        length = sizeof(sockaddr);
        if (getsockname(sock, (struct sockaddr *)&sockaddr, &length ) < 0 )
                        {
                perror("Error in getsockname()");
                exit(1);
        }
        printf("The socket tcp port # is  #%d\n", ntohs(sockaddr.sin_port));

/*
*               Fix the maximum number of clients waiting connection and
*               leave the server in passive mode
*/
                        listen(sock, 5);
                        puts("Server ready for client connection");
                        init = 1; /* end of initialization */

        case 1:  /* after initialization continue here */
                        newsock = accept(sock, (struct sockaddr *) 0, (size_t *) 0);
                        return newsock;
        }
}
/* ==== End of wait_connection ================================================ */



/* ***************************************************************************
*       Function recvlin() reading a socket connection with recv()
*       until newline (\n) or EOF is read. A flush of the remaining characters
*       is done until newline or EOF if the buffer is full. It returns a full
*       buffer.
*
*       conn: socket # (input)
*       *buff: address of the buffer receiving a complete line (input)
*       buffsz: buffer size in octets (input)
*  ==>  recvln returns the length in octets of read characters in buff
* ***************************************************************************
```

```
*/

int recvln(int conn, char *buff, int buffsz)
{
        char    *bp = buff, c;
        int     n;                      /* #octets read by recv() */

/*      While buffer not full and we can read characters
*       different from \n, octets are saved in buff
*/
        while (bp - buff < buffsz &&
            (n = recv(conn, bp, 1, 0)) > 0)
        {
                if (*bp++ == '\n') return (bp - buff);
        }

        if (n < 0)                      /* read has failed */
                return -1;

/*      If the buffer is full, ignore the rest until \n */
        if (bp - buff == buffsz)
                while (recv(conn, &c, 1, 0) > 0 && c != '\n');
        return (bp - buff);
}

/* ==== End de recvln ======================================================== */



/* ************************************************************************
*       Function send_header() sending the http 1.0 header to client
*       with the status, server name, content type and content length
*  ************************************************************************
*/

void send_header(int conn, int stat, int len)
{
        char    *statstr, buff[BUFFSIZE];

/*      change the status code to a character string message */

        switch(stat)
        {
        case 200:
                statstr = "OK";
                break;
        case 400:
                statstr = "Bad Request";
                break;
        case 404:
                statstr = "Not Found";
                break;
        default:
                statstr = "Unknown";
                break;
```

```
        }

/*
*       send the parameter server header
*       "HTTP/1.0 Server Content-Length Content-Type" to client
*/

        sprintf(buff, "HTTP/1.0 %d %s\r\n", stat, statstr);
        send(conn, buff, strlen(buff), 0);

        sprintf(buff, "Server: %s\r\n", SERVER_NAME);
        send(conn, buff, strlen(buff), 0);

        sprintf(buff, "Content-Length: %d\r\n", len);
        send(conn, buff, strlen(buff), 0);

        sprintf(buff, "Content-Type: text/html\r\n");
        send(conn, buff, strlen(buff), 0);

        sprintf(buff, "\r\n");
        send(conn, buff, strlen(buff), 0);
}
/* ===================== end of send_header ================================ */



/* ************************************************************************
*       Function send_eof() closes the connection from this end
*       by using a shutdown() call which shuts down all or part of a full-duplex
*       connection on the associated socket .
*       0 :     further receives will be disallowed,
*       l :     further sends will be disallowed (this mode is used),
*       2:      further sends and receives will be disallowed
* ************************************************************************
*/

void send_eof(int conn)
{
        shutdown(conn,1);
        return;
}

/*==================== end of send_eof ================================== */
```