COP4635 Sys & Net II **Project 1: HTTP Server** **Spring 2025**

HTTP Server

OVERVIEW

The objective of this project is to demonstrate the interactions between an HTML Web server with an HTML client. The code for the server program is to be written by the student. The HTML client that will communicate with the process created by running this server could be any internet browser program that you run on the local host. In a subsequent and separate project, you will eventually be writing the code for a client program to communicate with the server program that you code for this project, but that will be a separate project, and you don't need to do that for this project. The server will also have some web pages and other file/s that it will send to the browser client upon request by the client. This would demonstrate how the server interacts with the browser as a client, by transferring some elements in an HTML file in the server folder (if they are in another folder, the path should be accordingly resolved) and generating the standard error messages if the browser requests for a file that is not with the server.

THE PROGRAMS

The HTTP client-server system consists of two programs: a commonly available browser and a ***httpServer*** coded by you. The ***httpServer*** program serves HTML files to the browser acting as the http Client upon request from the client if it has the file/s requested. If it does not, it sends a standard "Not found" error message to the browser client. The client and server programs use **TCP** for communication.

IMPLEMENTATION

Program an HTTP server in C++. The server will listen on a valid port. Make sure you use a port number from the valid port number range (60001 – 60099). When a client makes a request for an HTML file using a valid HTTP/1.1 method like 'GET', the HTTP server will check if the file is a valid HTML file with the server. If the file is a valid HTML file, the server should send it to the client with proper HTTP header. If the file requested by the client is not a valid file with the server, or if the client request for the file is not recognized, the server should respond with a small, but descriptive error message that should be received by the client and displayed. Your HTTP server should be verbose and display all activity going on at the server.

Your server should respond to any standard browser program that is chosen to run as the client. If a user makes a request for the HTML page files while the server is running, the server should send the file to the browser if the request is valid. In case of the valid request/s the browser would receive the requested HTML file and display it on the screen.

To test the file access through the server, you have been provided with an **index.html** file that incorporates two hyperlinks. The first link points to an image object (also provided to you, but you could play around and add a different or more image files to this communication if you like) that gets displayed on the client browser by clicking the link. There is a second link on the **index.html** page that points to a non-existent html page and can help you test how non-existent pages are handled by the HTTP client-server system. Feel free to play around with these HTML files to test other aspects of web server and web client interactions.

Make sure that your HTTP server program does not terminate unless forced by the tester to stop. The server process should be coded by you to let the user request for more files in the same way through the browser client, and not exit immediately. If the server is still active and listening, it is supposed to respond accordingly. If the server has stopped working, this should also be indicated in the client program running. Make sure that you have provisions for handling both - normal and abnormal terminations of the server program, displaying appropriate codes and descriptive messages.

IMPLEMENTATION SUGGESTIONS

We suggest reading lecture notes and RFC for details on HTTP protocol and TCP networks discussed in class. You may also examine the reading material and the provided sample program on TCP programming. Read these pages thoroughly before starting this project. For your implementation you may need to use the following system calls (not an exhaustive list – and does not mean you have a compulsion to use only these) to create a socket, bind it, send and receive messages and perform file service.

socket()	connect()	gethostname()	bind()
read()	write()	close()	listen()
fflush()	sprintf()	getaddrinfo()	freeaddrinfo()
fscanf()	sendto()	recvfrom()	sigaction()

Some videos have also been posted to get you started with the project. You don't have to follow the system calls suggested or discussed in the videos. You might come across several ways and means to implement the project. If it lets you achieve all the objectives listed in the project document, feel free to use any other system calls as it is your project. What we have in the videos is just one possible way to get the project done.

WORKING IN TEAMS

Teams of up to two students can work on this project. If you wish to work in teams, you must make sure that you inform the instructor about your team ASAP by responding to the email from the instructor about this, and at least a week before the deadline. Last-minute information about any teams to the instructor will not be entertained and such groups will not be allowed.

DELIVERABLES & EVALUATION

Your project submission should follow the instructions below. Any submissions that do not follow the stated requirements will not be graded.

1. Follow the submission requirements of the instructor as published on eLearning under the Content area.

2. You should submit all the following files for this assignment:

a. source code files for the httpServer, and supporting code files if any
b. a single Makefile to compile the server program,
c. Relevant HTML and image files (already provided to you – please include them in your submission too, to make the grader's life easy)
d. a README file containing names of all project members, directions on compiling and running your files indicating usage and examples, and any other relevant details like an analysis if project wasn't completed, and what part was accomplished.
e. Screen captures of your running program showing all windows involved, in two or three stages of communication.
f. A protocol document (The protocol document must describe the protocols used by the system to make it work. At the minimum the document must describe the message exchange between the browser client and httpServer.)

Your program will be evaluated according to the steps shown below. Notice that the instructor/GA will not fix your syntax errors. However, they will make grading quick!

1. Program compilation with Makefile. The options –g and –Wall must be enabled in the Makefile. See the sample Makefile that I uploaded in eLearning.
   - If errors occur during compilation, the instructor/GA will not fix your code to get it to compile. The project will be given zero/partial points.
   - If warnings occur during compilation, there could be a deduction. The instructor/GA will test your code, though.
2. Program documentation and code structure.
   - The source code must be properly documented, and the code must be structured to enhance readability of the code.
   - Each source code file must include a header that describes the purpose of the source code file, the name of the programmer(s), the date when the code was written, and the course for which the code was developed.
3. Perform several test-runs with input of the grader's own choosing. At a minimum, the test runs should address the following questions.
   - Does the httpServer program compile properly and run, and is the communication between the server process and a standard browser process evident?
   - Can valid and invalid file links be tested with the browser as a client for your httpServer?
   - Is there ample and evident feedback on what is going on in the httpServer program while they are running, and when they exit? For example, when the server gets a connection request from the client, and when it proceeds to respond (or cannot respond) to a request does the server display what is going on in the server?

Keep in mind that documentation of source code is an essential part of computer programming. The better your code is documented, the better it can be maintained and reused. If you do not include comments in your source code, points will be deducted. You should refactor your code to make it more manageable and to avoid memory leaks. Points will be deducted if you don't refactor your code or if we encounter memory leaks in your program during testing.

DUE DATE

The project is due as indicated by the Dropbox for Project 1 in eLearning. Upload your complete solution to the dropbox. You can use the shared VM for creating and testing your code. Please mention in the comments at the top of your code if it tested perfectly with the shared VM and/or on the UWF-CS SSH server. Else, if you used WSL, please do that in the code as well as in the README file.

TESTING

Your solution needs to compile and run on the CS department's SSH server. I/the GA will compile and test your programs on the SSH server and a Linux VM. Therefore, to receive full credit for your work it is highly recommended that you test & evaluate your solution to make sure that we will be able to run your programs and test them successfully. For security reasons, most of the ports on the servers have been blocked from communication. Ports that are open for communication between the public servers' range in values from 60,000 to 60,099.

Some students have found it easier to develop utilizing WSL for the project rather than the ssh server and/or the provided VM image. If your program works in WSL please mention this in your README file and that is how we will grade them.

BONUS

Update the provided html file to be able to input text (whether it be through a text field, dropdown selection, etc). If your client can successfully send this text to the server, and the server acknowledges that it received this text, you will be provided **10 bonus points**.

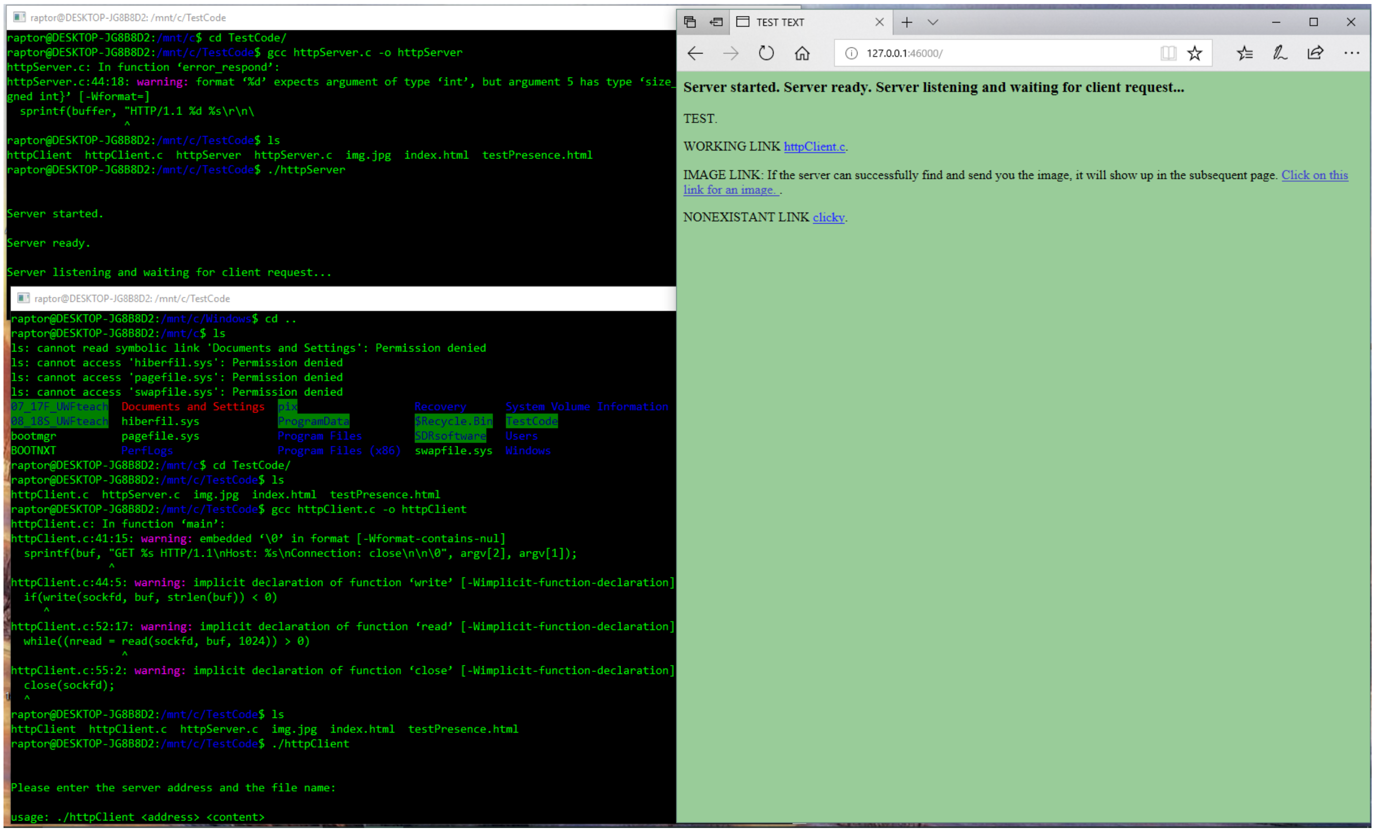
This bonus may be completed for either Project 1, Project 2, or both. If completed for Project 2, your client should be able to send text to the server and have the server acknowledge that this text was received. The bonus.

GRADING

This project is worth 100 points total. I will be updating this file with the rubric and will email you of this update. Please remember that there will be deductions if your code has memory leaks, crashes, has endless loops or is otherwise poorly documented or organized. We will not be able to grant any points for code that does not compile.

SAMPLE OUTPUTS

The following are some sample outputs from a bare-bones HTTP server and client code in C, as well as the interaction when the browser is used as a client. You are not writing the client code for this project, but you can see how interactions can be checked if the programs are coded in a verbose way. This code isn't as elaborate, efficient or informative as your project code should be. I used port address 46000 locally, for the test. The three screenshots show the server and client working together, then the client is terminated, and a standard browser is tested for requesting and displaying the valid and existent pages, followed by testing a non-existent file called **fake.html**, and finally testing the browser client with the server shut down.



```
raptor@DESKTOP-JG8B8D2:/mnt/c$ cd TestCode/
raptor@DESKTOP-JG8B8D2:/mnt/c/TestCode$ gcc httpServer.c -o httpServer
httpServer.c: In function ‘error_respond’:
httpServer.c:44:18: warning: format ‘%d’ expects argument of type ‘int’, but argument 5 has type ‘size_
gned int}’ [-Wformat=]
  sprintf(buffer, "HTTP/1.1 %d %s\r\n\
                  ^
raptor@DESKTOP-JG8B8D2:/mnt/c/TestCode$ ls
httpClient  httpClient.c  httpServer  httpServer.c  img.jpg  index.html  testPresence.html
raptor@DESKTOP-JG8B8D2:/mnt/c/TestCode$ ./httpServer


Server started.

Server ready.

Server listening and waiting for client request...
```

```
raptor@DESKTOP-JG8B8D2:/mnt/c/Windows$ cd ..
raptor@DESKTOP-JG8B8D2:/mnt/c$ ls
ls: cannot read symbolic link 'Documents and Settings': Permission denied
ls: cannot access 'hiberfil.sys': Permission denied
ls: cannot access 'pagefile.sys': Permission denied
ls: cannot access 'swapfile.sys': Permission denied
07_17F_UWFteach  Documents and Settings  pix                  Recovery      System Volume Information
08_18S_UWFteach  hiberfil.sys            ProgramData          $Recycle.Bin  TestCode
bootmgr          pagefile.sys            Program Files        SDRsoftware   Users
BOOTNXT          PerfLogs                Program Files (x86)  swapfile.sys  Windows
raptor@DESKTOP-JG8B8D2:/mnt/c$ cd TestCode/
raptor@DESKTOP-JG8B8D2:/mnt/c/TestCode$ ls
httpClient.c  httpServer.c  img.jpg  index.html  testPresence.html
raptor@DESKTOP-JG8B8D2:/mnt/c/TestCode$ gcc httpClient.c -o httpClient
httpClient.c: In function ‘main’:
httpClient.c:41:15: warning: embedded ‘\0’ in format [-Wformat-contains-nul]
  sprintf(buf, "GET %s HTTP/1.1\nHost: %s\nConnection: close\n\n\0", argv[2], argv[1]);
               ^
httpClient.c:44:5: warning: implicit declaration of function ‘write’ [-Wimplicit-function-declaration]
  if(write(sockfd, buf, strlen(buf)) < 0)
     ^
httpClient.c:52:17: warning: implicit declaration of function ‘read’ [-Wimplicit-function-declaration]
  while((nread = read(sockfd, buf, 1024)) > 0)
                 ^
httpClient.c:55:2: warning: implicit declaration of function ‘close’ [-Wimplicit-function-declaration]
  close(sockfd);
  ^
raptor@DESKTOP-JG8B8D2:/mnt/c/TestCode$ ls
httpClient  httpClient.c  httpServer.c  img.jpg  index.html  testPresence.html
raptor@DESKTOP-JG8B8D2:/mnt/c/TestCode$ ./httpClient


Please enter the server address and the file name:

usage: ./httpClient <address> <content>
```

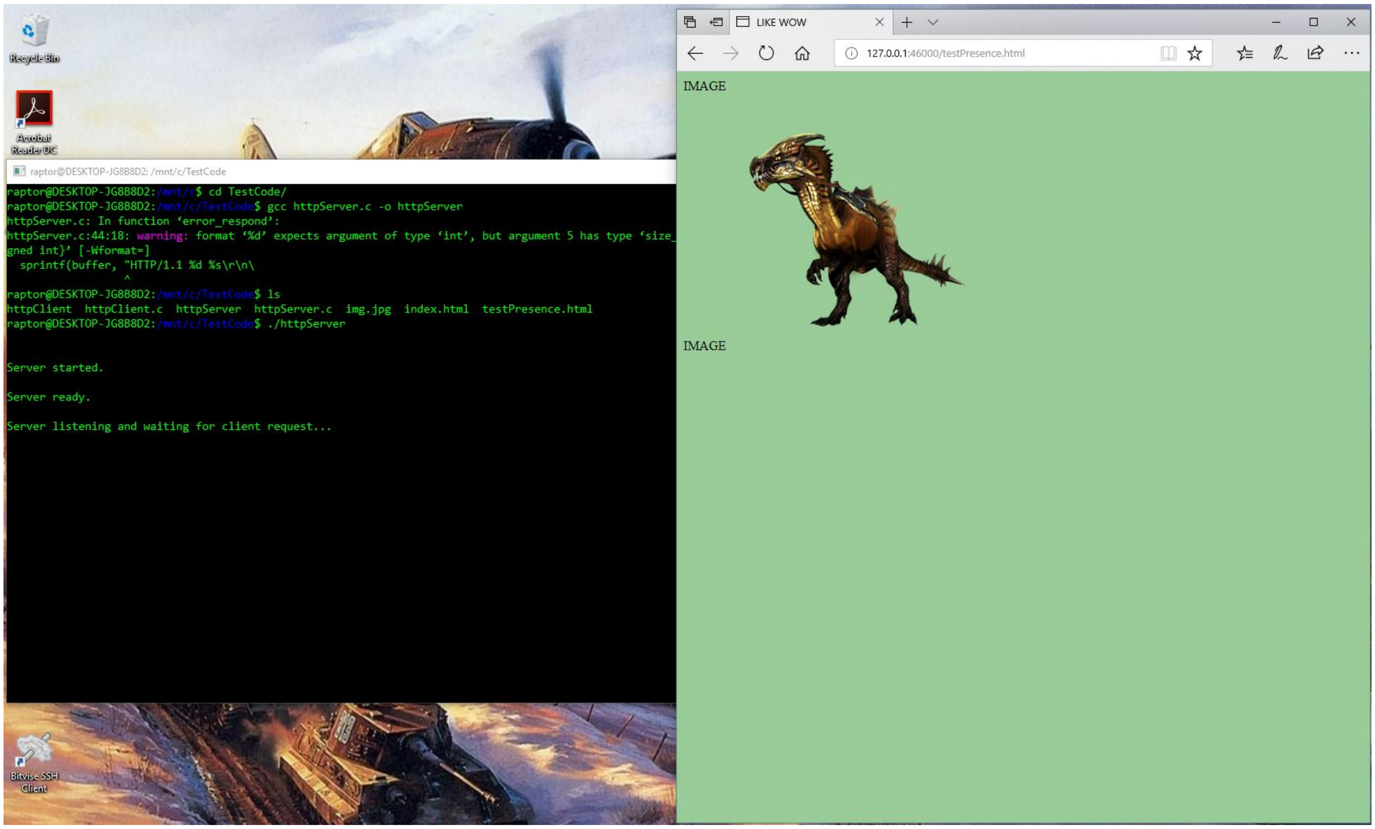
Server started. Server ready. Server listening and waiting for client request...

TEST.

WORKING LINK httpClient.c.

IMAGE LINK: If the server can successfully find and send you the image, it will show up in the subsequent page. Click on this link for an image. .

NONEXISTANT LINK clicky.

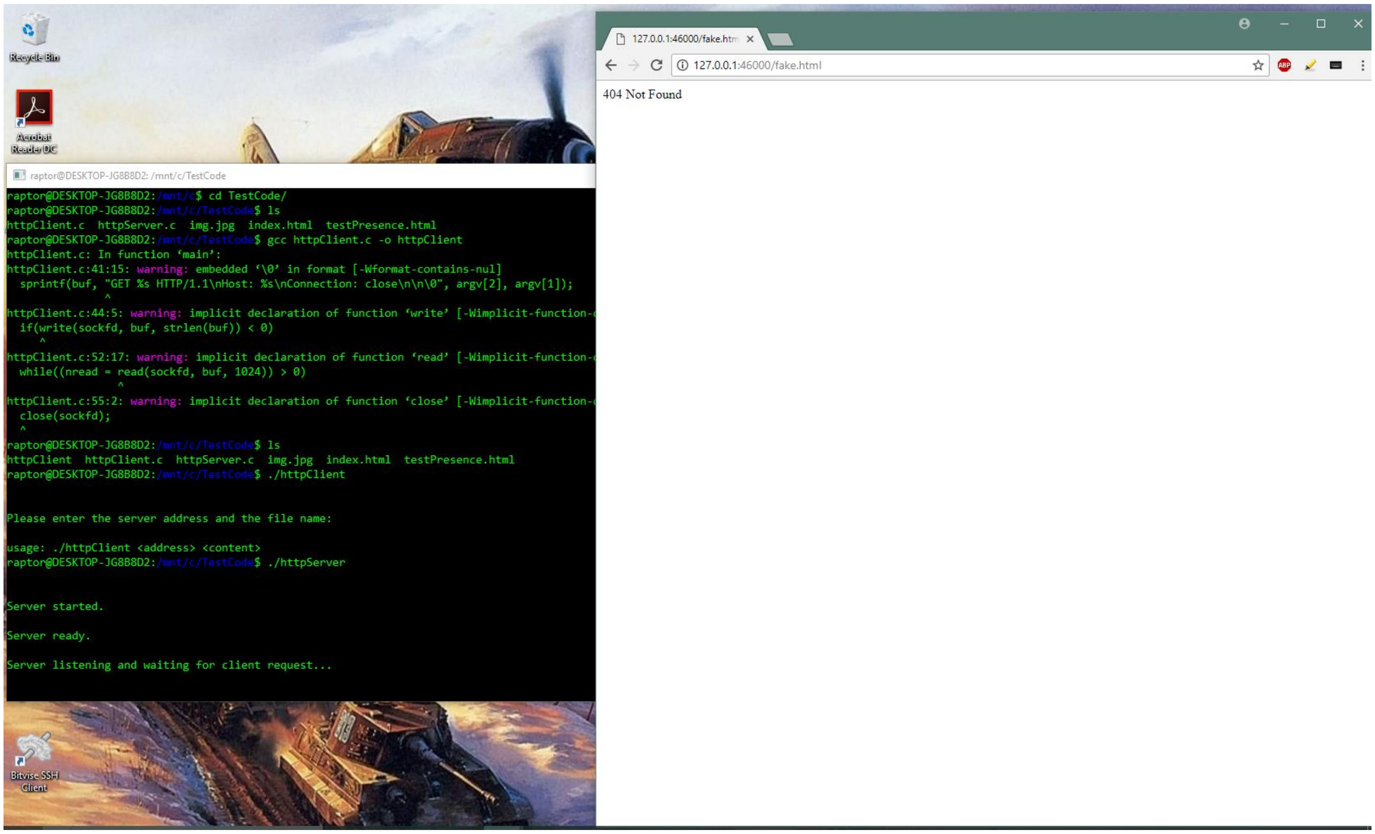


Recycle Bin
Acrobat Reader DC
raptor@DESKTOP-JG8B8D2: /mnt/c/TestCode
Bitvise SSH Client
LIKE WOW
127.0.0.1:46000/testPresence.html

```
raptor@DESKTOP-JG8B8D2:/mnt/c$ cd TestCode/
raptor@DESKTOP-JG8B8D2:/mnt/c/TestCode$ gcc httpServer.c -o httpServer
httpServer.c: In function ‘error_respond’:
httpServer.c:44:18: warning: format ‘%d’ expects argument of type ‘int’, but argument 5 has type ‘size_
gned int}’ [-Wformat=]
  sprintf(buffer, "HTTP/1.1 %d %s\r\n\
                  ^
raptor@DESKTOP-JG8B8D2:/mnt/c/TestCode$ ls
httpClient  httpClient.c  httpServer  httpServer.c  img.jpg  index.html  testPresence.html
raptor@DESKTOP-JG8B8D2:/mnt/c/TestCode$ ./httpServer


Server started.

Server ready.

Server listening and waiting for client request...
```

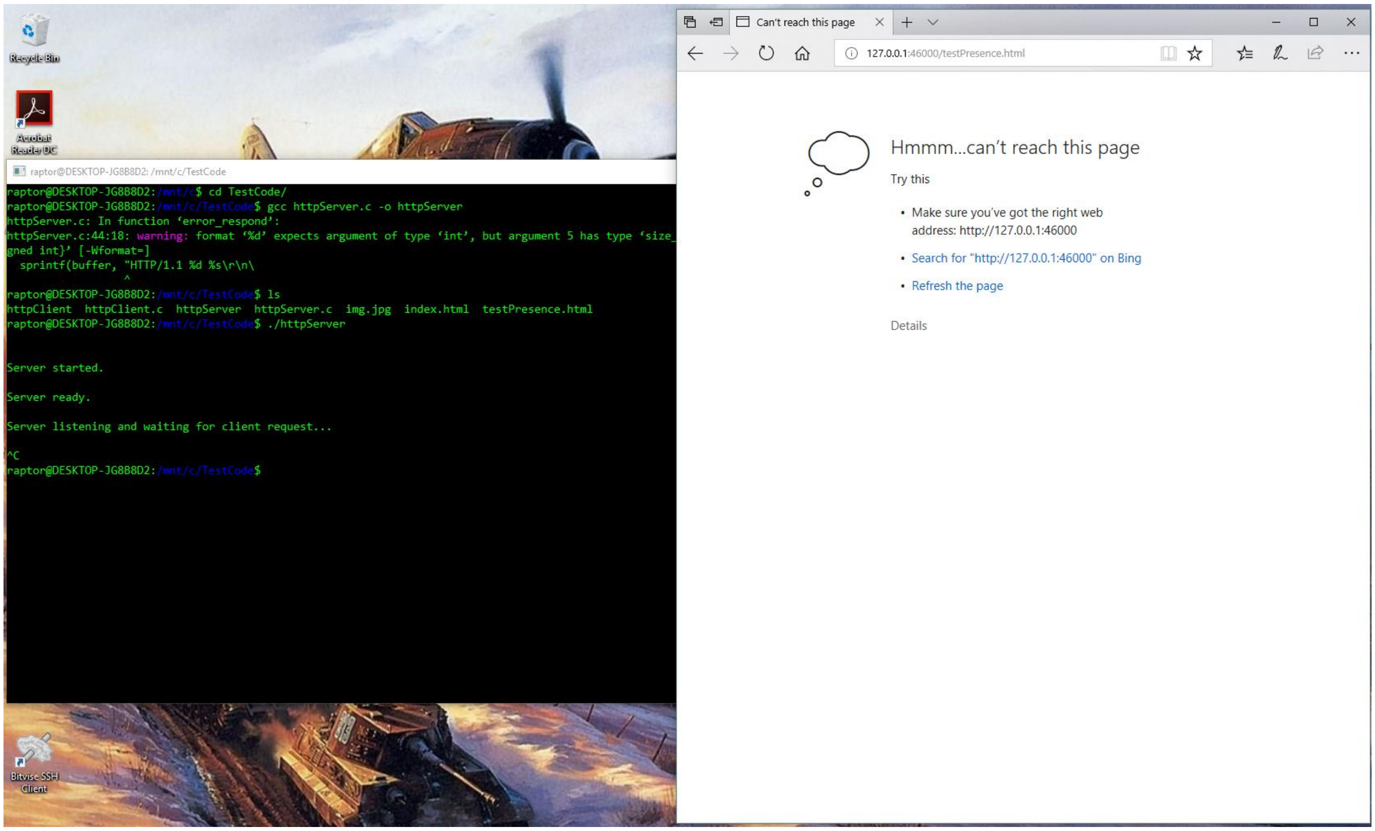
IMAGE

IMAGE



```
raptor@DESKTOP-JG8B8D2:/mnt/c$ cd TestCode/
raptor@DESKTOP-JG8B8D2:/mnt/c/TestCode$ ls
httpClient.c  httpServer.c  img.jpg  index.html  testPresence.html
raptor@DESKTOP-JG8B8D2:/mnt/c/TestCode$ gcc httpClient.c -o httpClient
httpClient.c: In function ‘main’:
httpClient.c:41:15: warning: embedded ‘\0’ in format [-Wformat-contains-nul]
  sprintf(buf, "GET %s HTTP/1.1\nHost: %s\nConnection: close\n\n\0", argv[2], argv[1]);
               ^
httpClient.c:44:5: warning: implicit declaration of function ‘write’ [-Wimplicit-function-
  if(write(sockfd, buf, strlen(buf)) < 0)
     ^
httpClient.c:52:17: warning: implicit declaration of function ‘read’ [-Wimplicit-function-
  while((nread = read(sockfd, buf, 1024)) > 0)
                 ^
httpClient.c:55:2: warning: implicit declaration of function ‘close’ [-Wimplicit-function-
  close(sockfd);
  ^
raptor@DESKTOP-JG8B8D2:/mnt/c/TestCode$ ls
httpClient  httpClient.c  httpServer.c  img.jpg  index.html  testPresence.html
raptor@DESKTOP-JG8B8D2:/mnt/c/TestCode$ ./httpClient


Please enter the server address and the file name:

usage: ./httpClient <address> <content>
raptor@DESKTOP-JG8B8D2:/mnt/c/TestCode$ ./httpServer


Server started.

Server ready.

Server listening and waiting for client request...
```

404 Not Found



Recycle Bin
Acrobat Reader DC
Bitvise SSH Client
raptor@DESKTOP-JG8B8D2: /mnt/c/TestCode
Can't reach this page
127.0.0.1:46000/testPresence.html

```
raptor@DESKTOP-JG8B8D2:/mnt/c$ cd TestCode/
raptor@DESKTOP-JG8B8D2:/mnt/c/TestCode$ gcc httpServer.c -o httpServer
httpServer.c: In function ‘error_respond’:
httpServer.c:44:18: warning: format ‘%d’ expects argument of type ‘int’, but argument 5 has type ‘size_
gned int}’ [-Wformat=]
  sprintf(buffer, "HTTP/1.1 %d %s\r\n\
                  ^
raptor@DESKTOP-JG8B8D2:/mnt/c/TestCode$ ls
httpClient  httpClient.c  httpServer  httpServer.c  img.jpg  index.html  testPresence.html
raptor@DESKTOP-JG8B8D2:/mnt/c/TestCode$ ./httpServer


Server started.

Server ready.

Server listening and waiting for client request...

^C
raptor@DESKTOP-JG8B8D2:/mnt/c/TestCode$
```

Hmmm...can’t reach this page

Try this

- Make sure you’ve got the right web address: http://127.0.0.1:46000
- Search for "http://127.0.0.1:46000" on Bing
- Refresh the page

Details