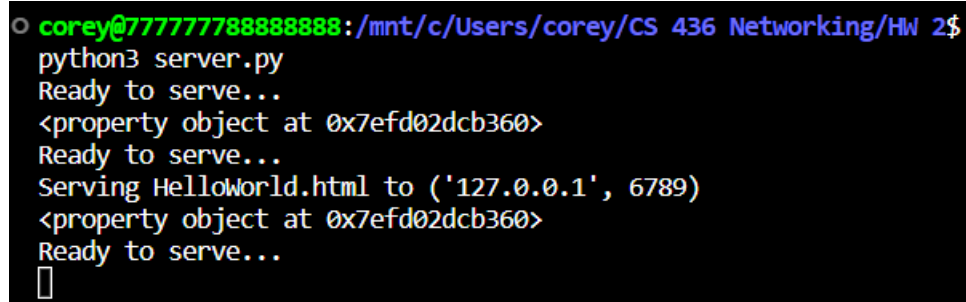


Homework 2 – HTTP Server

This project implements a simple Python HTTP server for TCP connections. The server includes basic security functionality to prevent access to certain files and directories. The server responds to HTTP requests by responding with a standard HTTP header and, if appropriate, the requested file. The project includes a few HTML files and a directory simulating a basic website for testing purposes.

Connection

The server communicates with a client over a non-persistent HTTP connection. The browser client initiates the connection, the sever accepts the connection, the client sends the request, the finally the server sends a response and closes the connection. The images that follow depict the request, response, and if any, the resource rendered in the browser window. Notably, the browser always makes a second request for a website favicon which is not included in the mock website used for this project. Therefore, that request is always returned with a “file not found” response.

A terminal window with a black background and white text. The prompt is 'corey@77777778888888:/mnt/c/Users/corey/CS 436 Networking/HW 2\$'. The user enters 'python3 server.py'. The output shows the server is ready to serve, receives a request for 'HelloWorld.html' from '127.0.0.1' on port 6789, and responds with a property object. The prompt returns to the shell.

```
corey@77777778888888:/mnt/c/Users/corey/CS 436 Networking/HW 2$  
python3 server.py  
Ready to serve...  
<property object at 0x7efd02dcb360>  
Ready to serve...  
Serving HelloWorld.html to ('127.0.0.1', 6789)  
<property object at 0x7efd02dcb360>  
Ready to serve...  
█
```

Figure 1: The server's perspective of a request.

Multithreaded Server

This implementation takes advantage of Python's threading library to enable the server to handle multiple requests simultaneously. When a connection is accepted, the server creates a new thread. The thread runs a function that contains the logic to parse the request and respond appropriately, and always closes the connection before the end of the thread's lifetime. The implementation ensures that every request is handled before the server is shutdown by forcing the main thread to wait to join every running thread before exiting.

Parsing the Request

The HTTP request message from the client includes the path of a resource as the second field in the first line. In this implementation, this is the only field of interest. The server attempts to open the file at this path. If an exception is thrown by the open function, the file is assumed to not exist and a “file not found” message is returned. Otherwise, an “OK” response is made.

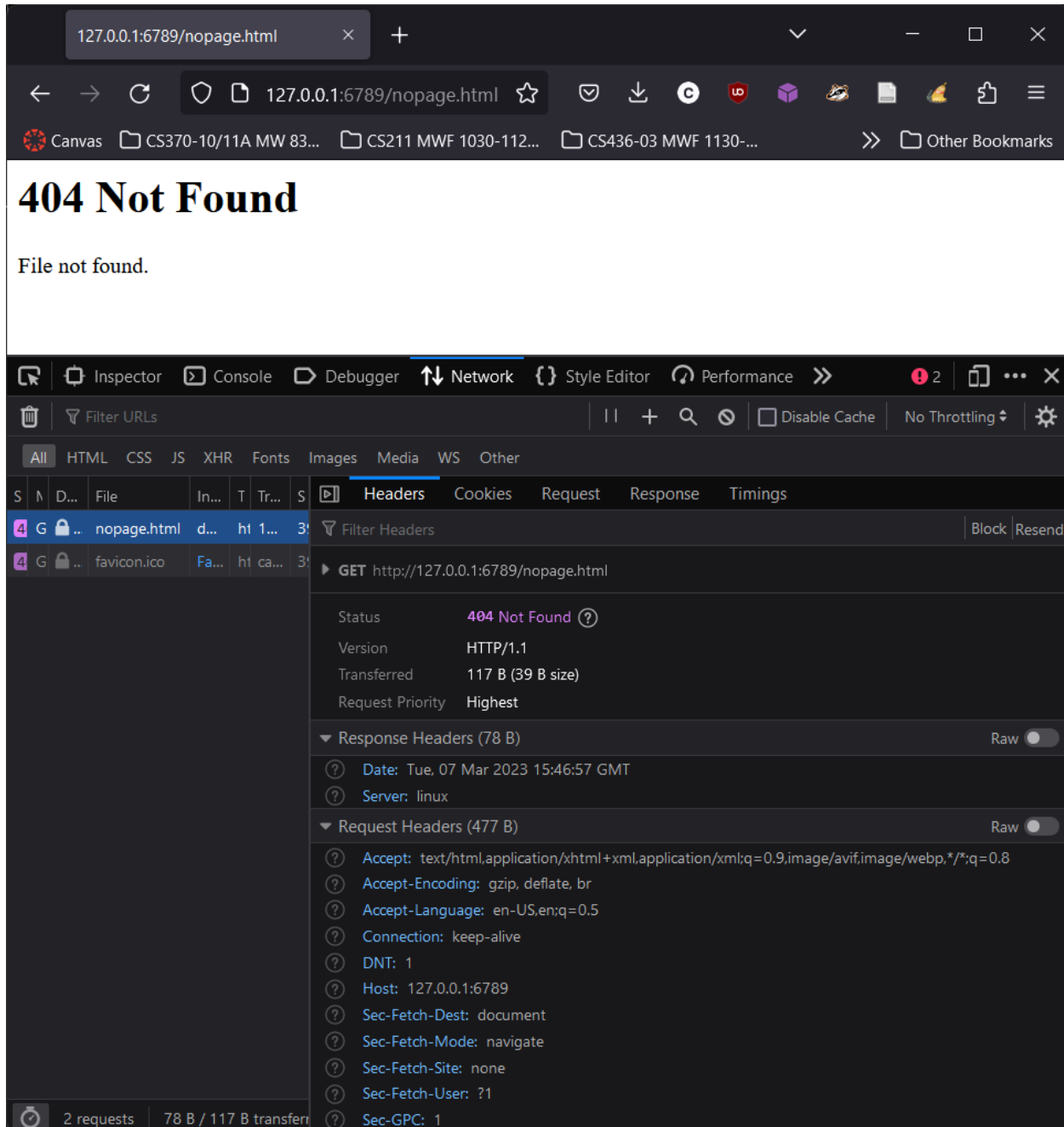


Figure 2: A request for a file that doesn't exist.

Security

Before the file is opened, the server checks if the requested resource is in the access-restricted directory “grades”. If so, the server responds with a 403 message, “forbidden”.

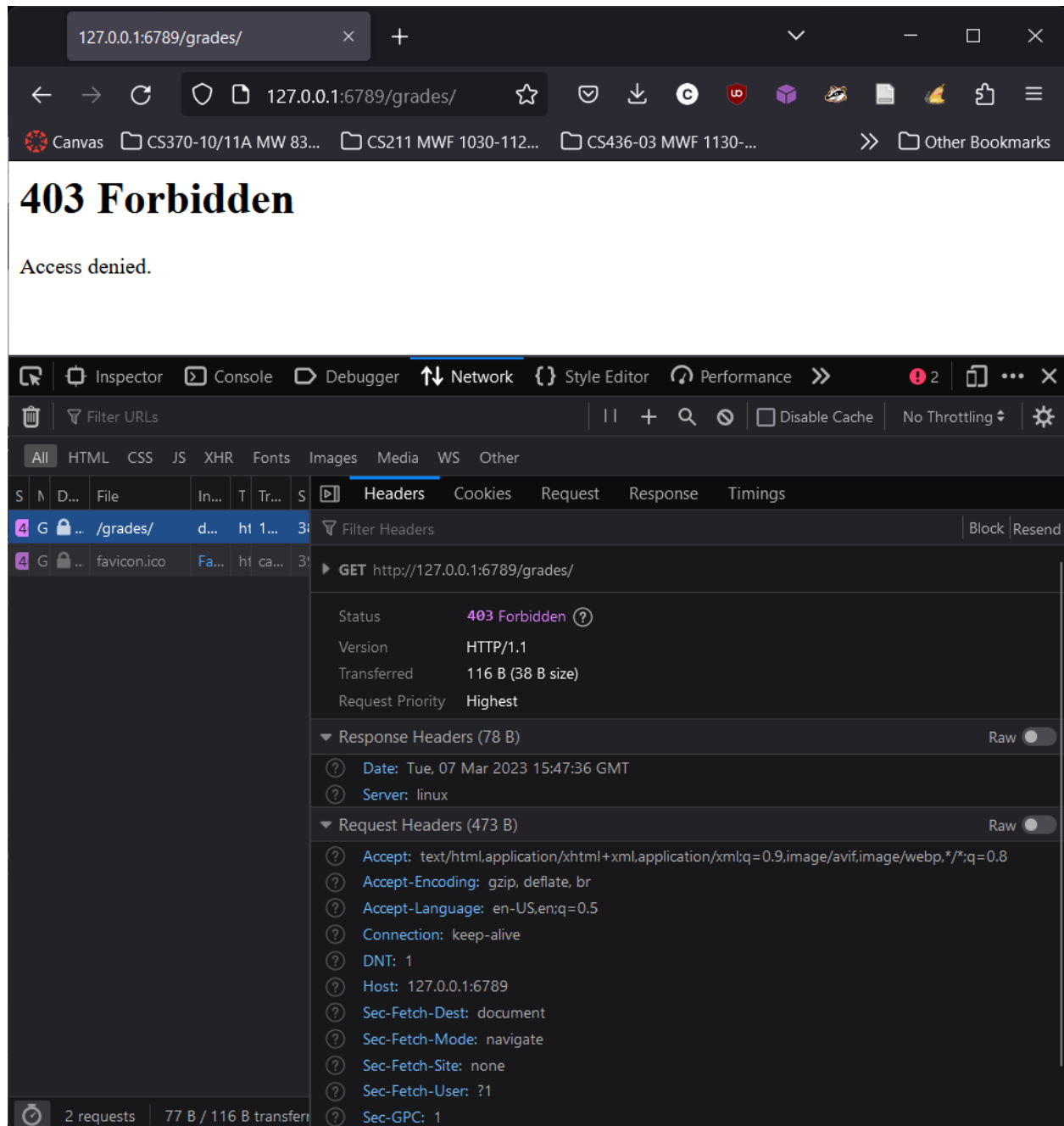


Figure 3: A disallowed request to list a directory.

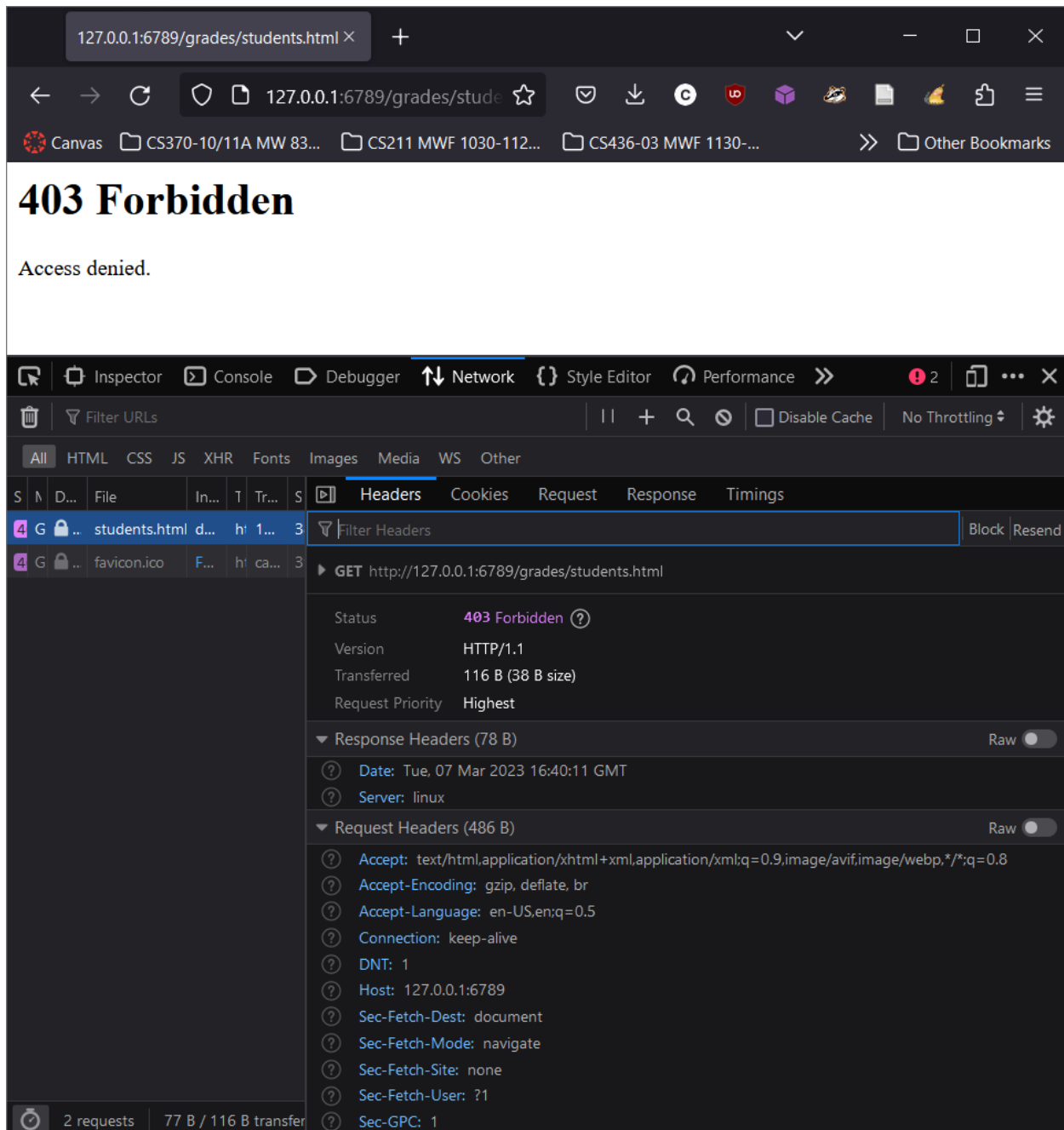


Figure 4: A request to access a forbidden file.

Constructing the Response

If the requested resource exists and access is not disallowed, in addition to the “OK” message in the HTTP header, the contents of the file desired by the client is included in the response.

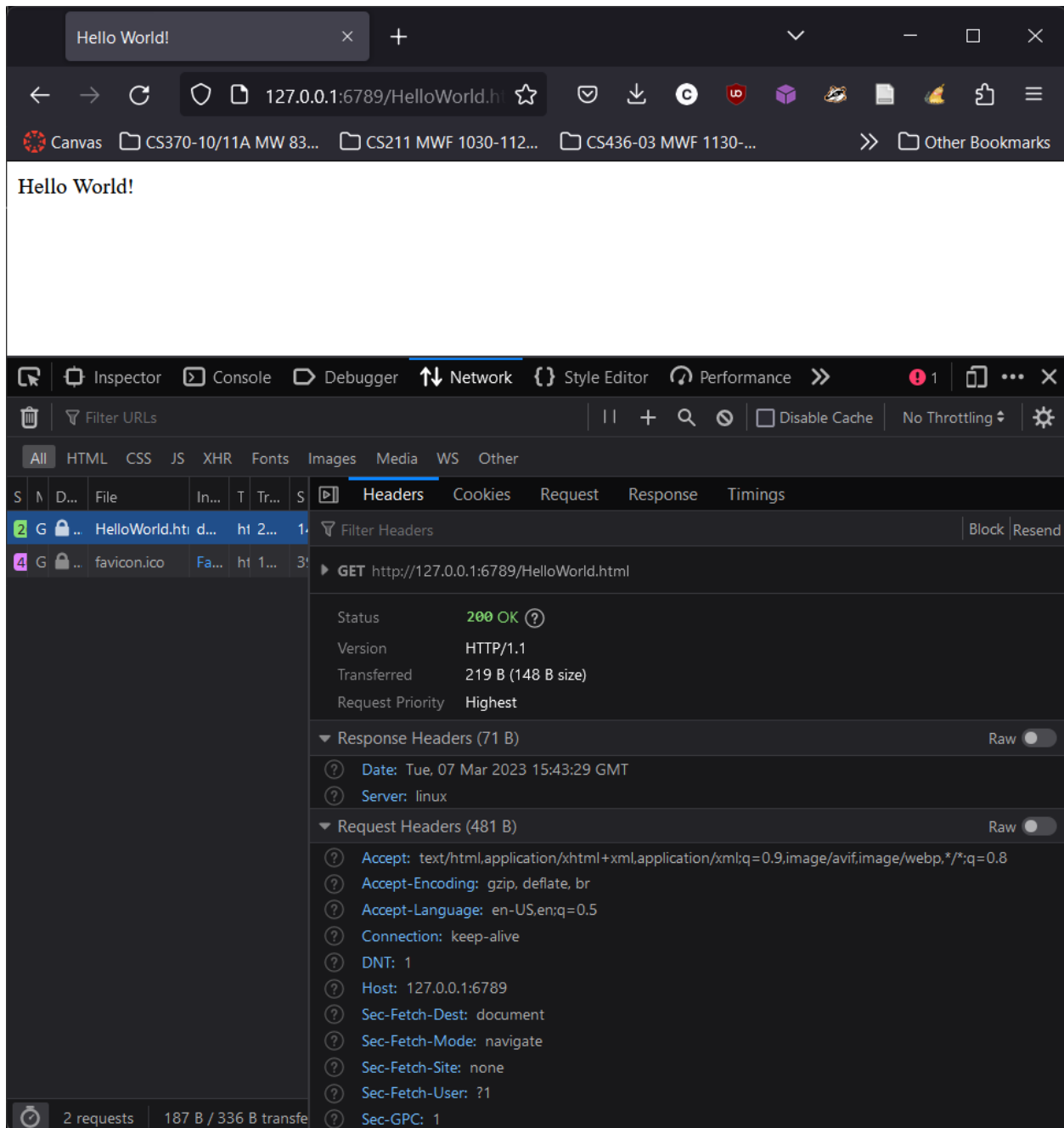


Figure 5: A successful request from the point of view of the browser.