CSC361: Tutorial 2

Go through P1 specifications

Demo with Nginx

Simple design idea

Objectives

- Use the STREAM socket (i.e., supported by TCP) in Python to create a Simple Web Server (SWS) with select() supporting both persistent and non-persistent HTTP connections.
  - you need to use select() and non-blocking socket() to complete P1 as a preparation for P2.

SWS requirements

- Support following HTTP request
  - GET /filename HTTP/1.0
- Support following HTTP header when supporting persistent HTTP connection
  - Connection: keep-alive
  - Connection: close
- If unsupported commands are received or in unrecognized format,
  - respond “HTTP/1.0 400 Bad Request”.
- If the file indicated by filename is inaccessible,
  - respond “HTTP/1.0 404 Not Found”
- For successful requests,
  - SWS will respond “HTTP/1.0 200 OK”

Example: HTTP Request

- HTTP Request = Request-line + Header-fields
- Request-line (Case-sensitive):
  - GET /filename HTTP/1.0
- Each header field consists of a case-insensitive field name followed by a colon (":"), optional leading whitespace, the field value, and optional trailing whitespace.
- Correct format:
  - “GET /filename HTTP/1.0\r\nConnection:Keep-alive\r\n\r\n”
  - “GET /filename HTTP/1.0\r\nConnection: keep-alive\r\n\r\n”
  - “GET /filename HTTP/1.0\r\n\r\n”
  - On Unix-like OS, “\n” = “\r\n”

Example: HTTP Response

- HTTP Response = Response-line + Header-fields
- Response
  - HTTP/1.0 400 Bad Request
  - HTTP/1.0 404 Not Found
  - HTTP/1.0 200 OK
- Header-fields (only to persistent connections)
  - Connection: Keep-alive

Example: 400 Bad Request

- Your server should return
  - “HTTP/1.0 400 Bad Request\r\n\r\n”

  when receiving the following requests:
  - “get /filename HTTP/1.0\r\n\r\n” Case-sensitive
  - “GET/filename HTTP/1.0\r\nConnection:Keep-alive\r\n\r\n” Space-sensitive
  - “go /filename HTTP\r\n\r\n” Words incorrect or missing

- For any bad requests, the connection will be immediately closed by your server regardless of the connection field in the request.

Example: 404 Not Found

- Your server should return
  - “HTTP/1.0 404 Not Found\r\nConnection: keep-alive\r\n\r\n”

when the request file does not exist and “Connection: Keep-alive” is in the request

- Your server should return
  - “HTTP/1.0 404 Not Found\r\nConnection: close\r\n\r\n”

when the request file does not exist and “Connection: Keep-alive” is not in the request

Example: 200 OK

- Your server should return
  - “HTTP/1.0 200 OK\r\nConnection: Keep-alive\r\n\r\n”

when the request file exists and “Connection: Keep-alive” is in the request

- Your server should return
  - “HTTP/1.0 200 OK\r\n\r\n”

when the request file exists and “Connection: Keep-alive” is not in the request

How to run SWS

- Open H2 in Piconet
  - python3 sws.py ip_address port_number
  - ip_address and port_number indicate where SWS binds its socket for incoming requests
- On H1 in PicoNet
  - nc sws_ip_address sws_port_number
  - type “GET /sws.py HTTP/1.0” followed by “Connection: keep-alive” and an empty line to request the file sws.py from SWS

SWS Output

- On H2 where you run sws:
  - For each served request, even if unsuccessfully, SWS will output a log line
  - "time: client_ip:client_port request, response",
  - e.g., "Wed Sep 15 21:44:35 PDT 2021: 192.168.1.100:54321 GET /sws.py HTTP/1.0; HTTP/1.0 200 OK".

- On H1 where you run nc:
  - In the aforementioned case, SWS shall keep the connection alive after sending back sws.py following an empty line after "Connection: keep-alive", and wait for the next request from the same client through the same TCP connection, until the connection times out, i.e., "Connection: close"
  - If the client does not include "Connection: keep-alive" or does include "Connection: close" in its request, SWS will close the connection after serving the request.

How to test SWS

1. 200 test: request an existing small-size file
2. 404 test: request a non-existing file
3. 400 test: send a bad request
4. Persistent connection test: send request with connection: keep-alive, and then send another request
5. Non-persistent connection test: send a request with connection: close
6. Large-file test: request a large-size file
7. Multi-client test: send persistent connection for a large-size file from h1, and send another connection from r
8. Timeout test: send a persistent connection and wait for 30s

How to check correctness

- Compare the returned messages with Nginx
- Nginx is a open-source web server
- The root directory for nginx is /var/www/html
- Put small and large files to this directory, then start nginx on h2 by command “service nginx start”

Simple design idea

```
import select
import socket
import sys
import queue
import time
import re

# Create a TCP/IP socket
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# set the socket to non-blocking mode
server.setblocking(0)
# bind address, server_address is defined by the input
server.bind(server_address)
# Listen for incoming connections
server.listen(5)
```

Create list to watch

```
# Sockets to watch for readability
inputs = [server]

# Sockets to watch for writability
outputs = []

# Outgoing message queues (socket:Queue)
response_messages = {}

# request message
request_message = {}
```

Handle readable sockets, if the readable socket is the server socket

```
while True:
    # Wait for at least one of the sockets to be ready for processing
        readable, writable, exceptional = select.select(inputs, outputs, inputs, timeout)
        for s in readable:
            if s is server, accept new connection, and append new connection socket
to the list to watch for readability
```

Handle readable sockets, if the readable socket is the connection socket

```
else:
    #Receive message from the receiving buffer
    message = s.recv(1024).decode()
    if message:
        #If message is not empty, we add the message to the queue for further process if format is correct
        request_message[s] =  request_message[s] + message
        if find “\r\n\r\n” or “\n\n” at the end of the message we can process the whole message:
            whole_message = request_message[s]
            #add connection socket s to the list for writable, as we will send back messages
            outputs,append(s)
            for each line of whole_message :
                if  format of the line is not correct:
                    response_messages{s} += “HTTP/1.0 400 Bad Request”
                else:
                    #Add response messages accordingly
```

Handle writable sockets

```
for s in writable :
        #get messages from response_message{s}
        try:
            next_msg = message_queues[s].get_nowait()
        except queue.Empty:
            #check if timeout or connection is persistent or not, and close socket accordingly
        else:
            #send messages and print logs if finish responding to a request
if s not in readable, writable, exception: #process timeout event
```

Note that we use request_message = {} to accumulate the request messages for each socket, and do not process the message until we reach the end of the request.

The reason is that for a connection, multiple requests can be arrived at the same time.

This feature is important as we will use an automatic tester which sends multiple requests simultaneously to evaluate your server

- To test your server whether this feature is enabled, you can create a text file (call test.txt) as follows:

GET /notfound HTTP/1.0

Connection:keep-alive

GET /hello.html HTTP/1.0

Connection:keep-alive

get /hello.html http/1.0

Connection:keep-alive

GET /notfound HTTP/1.0

Connection:keep-alive

- nc ip_address port < test.txt