Lab Number: Lab 1

Name: Anirudh Pal (pal5@purdue.edu)

PSO Instructor: Haleema Sadia

PSO Time: Monday @ 3:30 PM

Usage:

server.py

```
python server.py [port]
```

client.py

```
python client.py [IP] [port] [filename]
```

Server:

Listen:

The following code configures the socket.

```
# Make Socket Object
socketObj = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Bind to Port XXXXX
try:
    if CLARGS:
        socketObj.bind((HOST, int(sys.argv[1])))
    else:
        socketObj.bind((HOST, DPORT))
except socket.error as msg:
    if VERBOSE:
        print(str(msg))

# Listen for Connection
socketObj.listen(QSIZE)
```

Worker:

The worker accepts a connection and handles the request.

```
# Connection Handler
def worker():
    # Repeat for Ever
    while True:
        # Accept Connection
        conn, addr = socketObj.accept()
        if VERBOSE:
            print("Client Connection\nIP: " + str(addr[0]) + "\nPort: " +
str(addr[1]) + '\n')

        # Process Request
        parseHTTP(conn)
    conn.close()
```

Thread Pool:

This code block generates 5 threads that are used to handle up to five simultaneous connections.

```
## Create Thread Pool
threads = []
for i in range(TCOUNT - 1):
    # Associate Worker
    t = threading.Thread(target=worker)
    threads.append(t)

    # Kill if main killed
    t.daemon = True

    # Start
    t.start()

# Start
worker()
```

Parsing HTTP:

parseHTTp is used to parse the request and send the file. One assumption made is that the request will not exceed 4096 bytes. This can also be considered a security feature. The file is also broken up into 1024 block to be sent.

```
# Process HTTP Request
def parseHTTP(client):
    # Max Size (To Limit Size of Recieved Request)
    clientResponse = client.recv(1024)
    clientResponseDecoded = clientResponse
```

```
    # Print Response
    if VERBOSE:
        print("Client Data\n" + clientResponseDecoded + '\n')

    ## Get Path
    # Split Lines
    if VERBOSE:
        print("Data Lines and Words")
    i = 0;
    j = 0;
    path = ""
    for line in clientResponseDecoded.split('\n'):
        # Print Lines
        if VERBOSE:
            print("Line " + str(i) + ": " + line)
        i += 1

        # Split Words
        getBucket = False
        for word in line.split(' '):
            # Print Words
            if VERBOSE:
                print("Word " + str(j) + ": " + word)
            j += 1

            # Use Get Bucket
            if getBucket:
                path = word
                if VERBOSE:
                    print("Bucket Caught: " + path + '\n')
                break

            # Set Get Bucket
            if word == RTYPE:
                if VERBOSE:
                    print("Hit a GET\n")
                getBucket = True

        # Why Work Harder
        if path:
            break

    ## Process Path
    # Found Path
    if path:
        # Get Filename
        if path == '/':
            path = "Upload/index.html"
        else:
            path = "Upload/" + path[1:]
        if VERBOSE:
            print("Filename: " + path + '\n')

    # No Path
    else:
        # Error
        send400(client)
```

```
        if VERBOSE:
            print("parseHTTP(): No Path\n")
        return

    ## Process Filename
    try:
        file = open(path, "rb")
        sendHTTP(client, file, path)
        file.close()
    except:
        # Error
        send404(client)
        if VERBOSE:
            print("parseHTTP(): Not Real Path\n")
        return
```

Errors:

send404 and send400 are used to handle errors.

```
# Handle Errors
def send404(to):
    to.send("HTTP/1.1 404 Not Found\r\n\r\n")
def send400(to):
    to.send("HTTP/1.1 400 Bad Request\r\n\r\n")
```

Client:

The following code configures the socket.

```
# Make Socket Object
socketObj = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connect to Server
try:
    if CLARGS:
        socketObj.connect((sys.argv[1], int(sys.argv[2])))
    else:
        socketObj.connect((HOST, DPORT))
except socket.error as msg:
    if VERBOSE:
            print(str(msg))
```

sendHTTP is used to send a request and recieve and fill the file. The way I handle distinguishing the aknowledgement and the file content can be improved. I also except and write the file in blocks of 1024.

```
# Process Request
```

```
def sendHTTP(server, filename):
    # Build and Send Request
    request = "GET /" + filename +" HTTP/1.1\r\n\r\n"
    server.send(request)
    if VERBOSE:
        print("Request: " + request + '\n')

    # Parse Resonse
    result = server.recv(1024)
    if VERBOSE:
        print("Response: " + result + '\n')
    if result.find("HTTP/1.1 200 Document follows") == -1:
        if VERBOSE:
            print("Bad Response\n")
        return

    # Parse File
    if VERBOSE:
        print("Getting File")
    file = open("Download/" + filename, "wb")
    file.write(result[(result.find("\r\n\r\n") + len("\r\n\r\n")):])
    buf = server.recv(1024)
    while buf:
        file.write(buf)
        buf = server.recv(1024)
    file.close()
    if VERBOSE:
        print("Got File")\

# Send Request
sendHTTP(socketObj, FILE)
```
