Super HTTP - Buffering

Welcome to the real world, where HTTP requests don't necessarily fit into one call to recv(), either because

1. They're too big
2. The individual packets don't come all at once

Problem #1 - Big requests

1. Continue with the same HTTP server `superhttp.py`. Make sure files uploaded via `/upload` are uploaded to file `uploads/userfile.txt`
2. As before, try to upload a small text file `note.txt` using curl: `curl -X POST 127.0.0.1:8005/upload -d @note.txt`
3. Now, create a file `bignote.txt` that contains 250,000 characters (you can use python to create it), and try to upload the big file. Look in the upload directory and notice the file doesn't upload completely - you can see by the size. This is a problem!

Problem #2 - Split requests

1. In the previous exercise, we sent an entire request at once with the command `type request.txt | nc 127.0.0.1 8005`. It works because the entire request is returned from the call to recv().
2. Now, connect to the server with netcat: `nc --crlf 127.0.0.1 8005` (notice the flag `--crlf` - to make the newlines be windows-style '\r\n' newlines, as expected in the HTTP protocol)
3. Type a simple HTTP request as follows (in under 10 seconds, because of the timeout we added previously)

```
GET /abcd HTTP/1.1
```

1. Notice that the request should have two newlines, but after hitting 'Enter' for the first time, the server's call to recv() will already return with an incomplete request.
2. You can check in Wireshark - when you press 'Enter' in netcat, it sends a TCP packet. This makes recv() finish with the data. But an HTTP request isn't complete after just one 'Enter'. This is a problem!

Instructions

1. Create a function named `get_client_request(client_socket)` that is responsible for receivbing the full HTTP request from the client before it is parsed.

- Fix both issues by buffering the request by calling recv() in a loop before parsing it.
- The server should handle the request as soon as it is received.
- Think what the ending condition of the loop should be; or in other words, how would you know when the request has been read in its entirety. There could be different conditions for each of the above problems.
- Also notice that now the configuration value 'max*request*size' shouldn't be compared against the length of a single result of recv(), but instead against the buffer - every time it is appended to.

Note

Replace this code:

```
request =
client_socket.recv(config['max_request_size']).decod
e()
if not request:
    raise ConnectionResetError()
```

With a call to your new function:

```
request = get_client_request(client_socket)
```

To submit

Submit file `superhttp.py`.