1 Introduction

This project requires understanding of a number of topics covered in this class. Among them:
laying as an approach for management of complex systems, as illustrated by the Internet network architecture; enforcing the modularity, illustrated by the client-server paradigm; naming, illustrated by naming in the Internet; enforcement of consumer-producer rules, and so on.

We first review some basic ideas regarding communication in the Internet, then illustrate the client-server paradigm, and finally give an example of a multi-threaded server.

2 Communication in the Internet

Figure 1 illustrates the Internet protocols.

**IP address** serves two functions: host identification and location addressing. All communication in the Internet must use the IP protocol. The IP addresses are used by the IP protocol to route messages from source to the destination through the Internet.

IPv4 uses 32-bit addresses; the address space is limited to 4,294,967,296 (2^{32}) possible unique addresses. Addresses for special purposes: private networks (18 million addresses); multicast addresses (270 million addresses). Addresses represented in dot-decimal notation e.g., 218.96.17.12).

IPv6 uses 64-bit addresses; the address space is limited to 2^{64} possible unique addresses.

Figure 2 illustrates message delivery to hosts and processes.

3 The Client-Server Paradigm

Figure 3 illustrates the communication supporting a client-server paradigm.

4 Communication in the Internet

**IP address** serves two functions: host identification and location addressing. All communication in the Internet must use the IP protocol. The IP addresses are used by the IP protocol to route messages from source to the destination through the Internet.

IPv4 uses 32-bit addresses; the address space is limited to 4,294,967,296 (2^{32}) possible unique addresses. Addresses for special purposes: private networks (18 million addresses); multicast addresses (270 million addresses). Addresses represented in dot-decimal notation e.g., 218.96.17.12).

IPv6 uses 64-bit addresses; the address space is limited to 2^{64} possible unique addresses. No flag day
5 The World Wide Web is a well known application of the client-server model

The Hypertext Transfer Protocol (HTTP) is an application protocol used by the World Wide Web. Berners-Lee and his team are credited with inventing the original HTTP protocol along with the HTML and the associated technology for a web server and a text-based web browser. The first version of the protocol had only one method, namely GET, which would request a page from a server. The response from the server was always an HTML page.

The client submits an HTTP request message to the server. The server, which stores content, or provides resources, such as HTML files, or performs other functions on behalf of the client, returns a response message to the client. A response contains completion status information about the request and may contain any content requested by the client in its message body. Figure 4 shows multiple HTTP clients and a multi-threaded HTTP server as well as the HTTP request codes and the HTTP response codes.

At request of several students I provide an example of Java code code for a multi-threaded HTTP server; the code is due to Jacob Jenkov. When the multi-threaded server with the code presented next is running it can be accessed using any browser by typing the address http://localhost:9000/.
6 The code for a simple single-threaded HTTP server

The main loop is the critical component: the server waits for a request, processes it and waits for the next request.

```java
package servers;

import java.net.ServerSocket;
import java.net.Socket;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

public class SingleThreadedServer implements Runnable{

    protected int serverPort = 9000;
    protected ServerSocket serverSocket = null;
    protected boolean isStopped = false;
    protected Thread runningThread = null;

    public SingleThreadedServer(int port){
        this.serverPort = port;
    }

    public void run(){
        synchronized(this){
            this.runningThread = Thread.currentThread();
        }
        openServerSocket();

        while(! isStopped()){  
            Socket clientSocket = null;
            try {
                clientSocket = this.serverSocket.accept();
            } catch (IOException e) {
                if(isStopped()) {
                    System.out.println("Server Stopped.");
                    return;
                }
                throw new RuntimeException("Error accepting client connection", e);
            }
            try {
                processClientRequest(clientSocket);
            } catch (IOException e) {
                //log exception and go on to next request.
            }
        }
        System.out.println("Server Stopped.");
    }
}
```
private void processClientRequest(Socket clientSocket)
    throws IOException {
    InputStream input = clientSocket.getInputStream();
    OutputStream output = clientSocket.getOutputStream();
    long time = System.currentTimeMillis();
    output.write(("HTTP/1.1 200 OK
      \n<html><body>
        Singlethreaded Server: " +
        time +
        "</body></html>\n").getBytes());
    output.close();
    input.close();
    System.out.println("Request processed: " + time);
}

private synchronized boolean isStopped() {
    return this.isStopped;
}

public synchronized void stop(){
    this.isStopped = true;
    try {
        this.serverSocket.close();
    } catch (IOException e) {
        throw new RuntimeException("Error closing server", e);
    }
}

private void openServerSocket() {
    try {
        this.serverSocket = new ServerSocket(this.serverPort);
    } catch (IOException e) {
        throw new RuntimeException("Cannot open port 8080", e);
    }
}

7 The code to run this example

SingleThreadedServer server = new SingleThreadedServer(9000);
new Thread(server).start();

try {
    Thread.sleep(10 * 1000);
} catch (InterruptedException e) {

8 The code for a multi-threaded server

A multithreaded server instead of processing an incoming request in the same thread which accepts client connections, hands off the connection to a worker thread to process the request. Thus, the thread listening for incoming requests spends as much time as possible in the serverSocket.accept() call; the risk for clients being denied access to the server because the listening thread is not inside the accept() call is minimized. The main loop for a multithreaded HTTP server is different

```java
while(! isStopped()){
    Socket clientSocket = null;
    try {
        clientSocket = this.serverSocket.accept();
    } catch (IOException e) {
        if(isStopped()) {
            System.out.println("Server Stopped.");
            return;
        }

        new Thread(
            new WorkerRunnable(
                clientSocket, "Multithreaded Server"
            ).start();
    }
}
```

9 The code for the WorkerRunnable class passed to the worker thread constructor

```java
package servers;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.IOException;
import java.net.Socket;

public class WorkerRunnable implements Runnable{
    protected Socket clientSocket = null;
    protected String serverText = null;
```
public WorkerRunnable(Socket clientSocket, String serverText) {
    this.clientSocket = clientSocket;
    this.serverText = serverText;
}

public void run() {
    try {
        InputStream input = clientSocket.getInputStream();
        OutputStream output = clientSocket.getOutputStream();
        long time = System.currentTimeMillis();
        output.write(("HTTP/1.1 200 OK\n\nWorkerRunnable: " +
                this.serverText + " - " +
                time +
                "\n").getBytes());
        output.close();
        input.close();
        System.out.println("Request processed: " + time);
    } catch (IOException e) {
        //report exception somewhere.
        e.printStackTrace();
    }
}
}
Figure 1: Internet family of protocols; multiplexing and de-multiplexing
IP address = (NetworkId, HostId)

Message delivery to processes

Sockets and ports

Figure 2: Message delivery to hosts and to processes
Figure 3: The Client-Server paradigm. (a) Request-response communication. (b) Connection establishment for HTTP. HTTP used TCP as a transport protocol.
An HTTP request contains one of the following methods:

- **GET** - get a resource
- **HEAD** - verify the link and conditions of a resource
- **POST** - input to a resource, usually a CGI script
- **PUT** - store a resource at the server
- **DELETE** - delete a resource
- **TRACE** - include all headers in a response

Sample HTTP status code in a response:

- 100 - Continue
- 200 - OK
- 205 - Reset Connection
- 301 - Moved Permanently
- 402 - Payment Required
- 404 - Not Found
- 405 - Method Not Allowed
- 407 - Proxy Authentication Required
- 415 - Unsupported Media Type
- 500 - Internal Server Error
- 504 - Gateway Timeout
- 505 - HTTP version Not Supported

Figure 4: HTTP clients and a multi-threaded HTTP server.