COMP 4621
Tutorial #3

Spring 2015

More on HTTP/TCP Programming

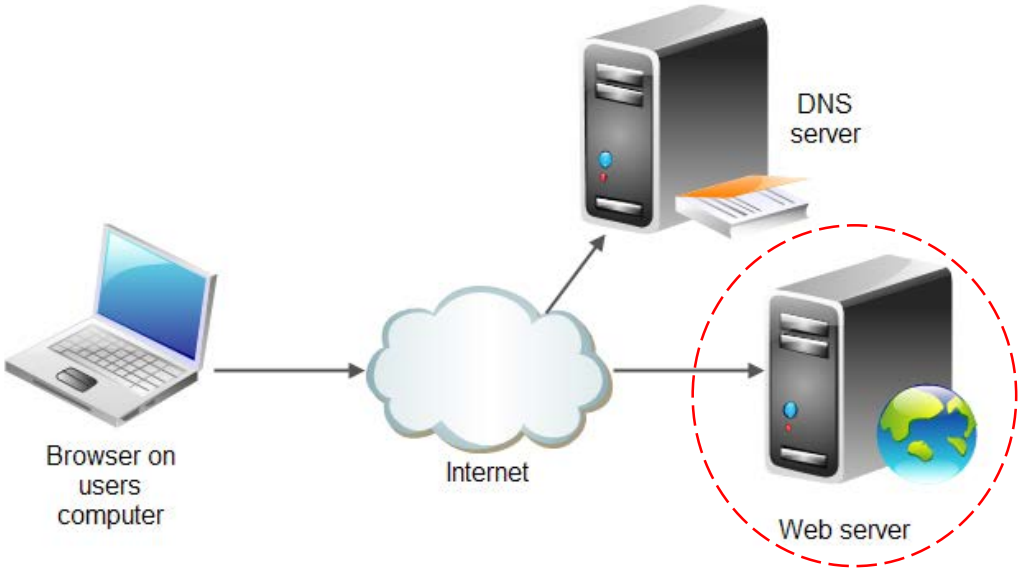
- A simple web server
- Multi-thread programming
- Web server with multi-threading

More on HTTP/TCP Programming

- A simple web server
- Multi-thread programming
- Web server with multi-threading

Web Server

- Web server is a **computer program** which:
  - responsible for accepting HTTP requests from clients (user agents such as web browsers),
  - and serving them HTTP responses along with optional data contents, which usually are web pages such as HTML documents and linked objects



Web Server Software

Vendor	Product	Web Sites Hosted	Percent
Apache	Apache	91,068,713	50.24%
Microsoft	IIS	62,364,634	34.4%
Google	GWS	10,072,687	5.56%
lighttpd	lighttpd	3,095,928	1.71%
nginx	nginx	2,562,554	1.41%
Oversee	Oversee	1,938,953	1.07%
Others	-	10,174,366	5.61%
Total	-	**181,277,835**	**100.00%**

APACHE
HTTP SERVER

Windows
IIS 8.5

Google Web Server

NGINX
http server

A Simple Web Server

- Today, we will play with a very sample web server:
  - Keep listening on a given port
  - Accept incoming connection
  - Understand users request
  - Response with a sample HTML webpage
- Give you the basic idea of what a web server is.

Web Server: Example

```
package lab3;
import java.net.*;
import java.io.*;

public class WebServer {
        public static void main(String[] args) throws IOException {
        // create server socket
        int nPort = 8090;
        ServerSocket svrSocket = new ServerSocket(nPort); // run the server;
        // keep listening on port 8090, serve for each req
        Socket socket = null;
```

Web Server: Example

```
while (true) {
        System.out.println("Server is ready for requst...");
        // block until TCP connection is established
        socket = svrSocket.accept();
        // Get the HTTP request content
        BufferedReader inFromClient = new BufferedReader(new InputStreamReader(clientSocket.getInputStream() ) );
        char reqBuf [] = new char [1024];
        inFromClient.read(reqBuf);
        String strReq = new String(reqBuf);
        strReq.trim();
        System.out.println("Req: \n"+strReq);
        // what should we do here? parse req!
        String [] arrHttpContent =  strReq.split(" ");
        String strMethod = arrHttpContent[0];
        String strTarget = arrHttpContent[1];

        if (strTarget.equals("//favicon.ico") ) {
                System.out.println("browser is requesting icon, forget it!");
                continue;//do nothing }
```

Web Server: Example

```
// send resp to client
String content = "welcome to my homepage";
String html = "HTTP/1.1 200 OK\n Connection:close\n Date: Mon, 23 Feb 2009 14:23:00 GMT\n"
+ "Server:Apache/1.3.0 (unix)\n Content-Length:" + content.length() + "\n“ + "Content-Type: text/html\n\n“ + content;
DataOutputStream outToServer = new DataOutputStream(socket.getOutputStream());
outToServer.writeBytes(html); // send out html
// clear up
outToServer.flush();
socket.close();
System.out.println("Resp has been sent."); } } }
```

A Simple HTML

```
<html>
        <head><title>Test of COMP 4621 Lab 3</title></head>
        <body>
                <center>
                <font size="10">welcome~<br>this is a simple webpage
for COMP 4621 lab 3</font>
                </center>
        </body>
</html>
```

Web Server: Practice

- Compile and run the example
- Open http://localhost in your browser
  - Can not open? Try http://localhost:8090, why?

- Practice
  a) Change the HTML content
  b) Print out the full HTTP reqest made by your web browser, see what is in there
  c) Try different browsers (if you have multiple browsers on your machine)
  d) When will a persistent HTTP be closed?

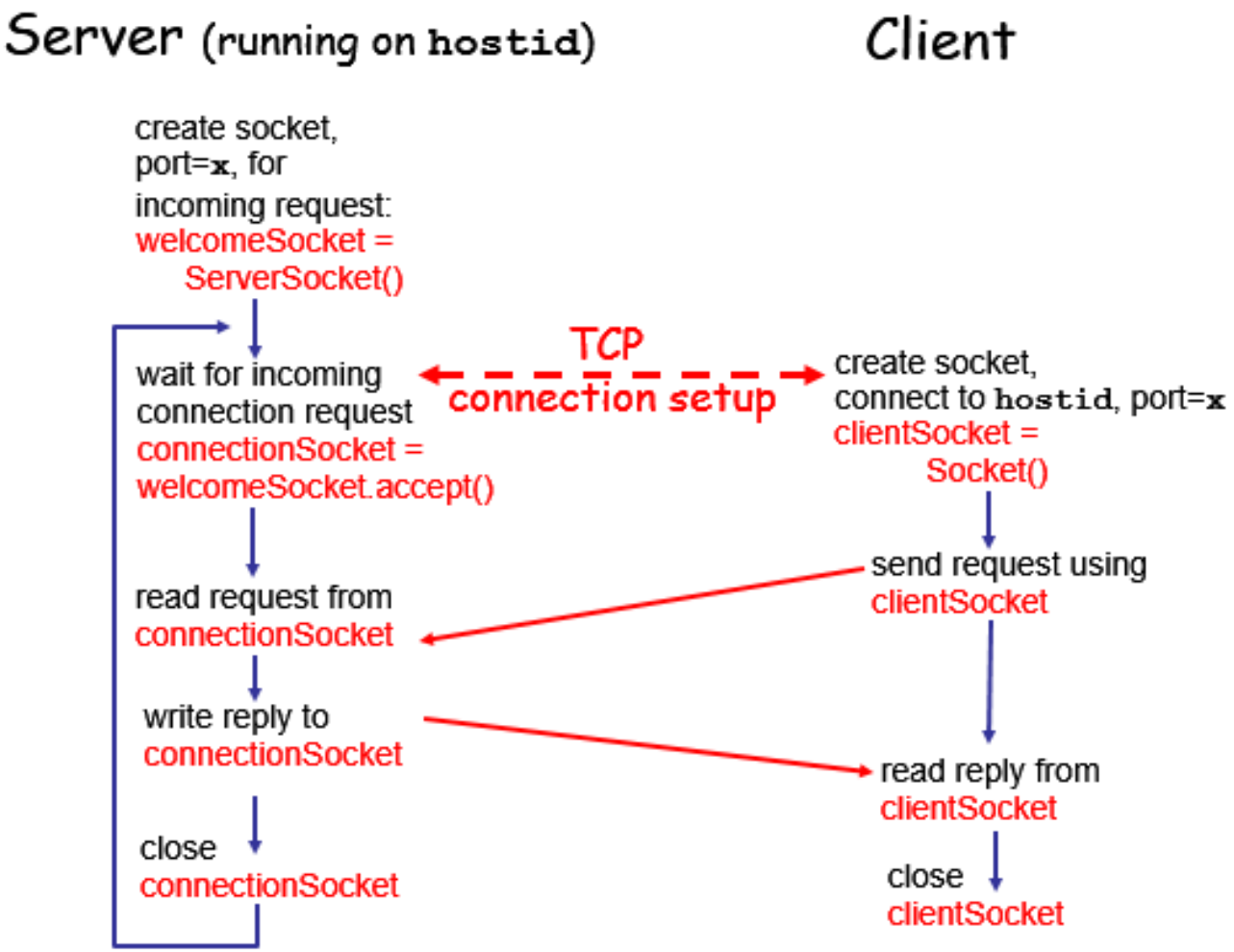
More on HTTP/TCP Programming

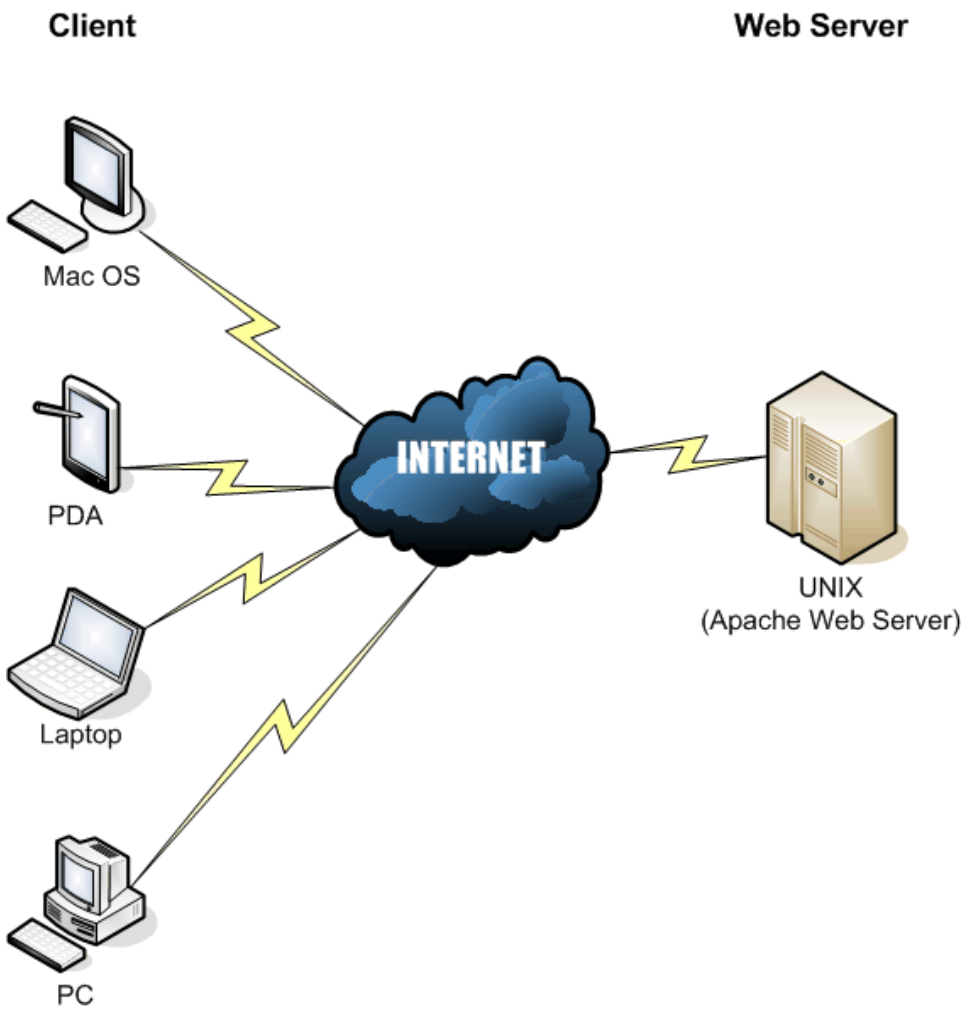
- A simple web server
- Multi-thread programming
- Web server with multi-threading

Multi-threading Programming

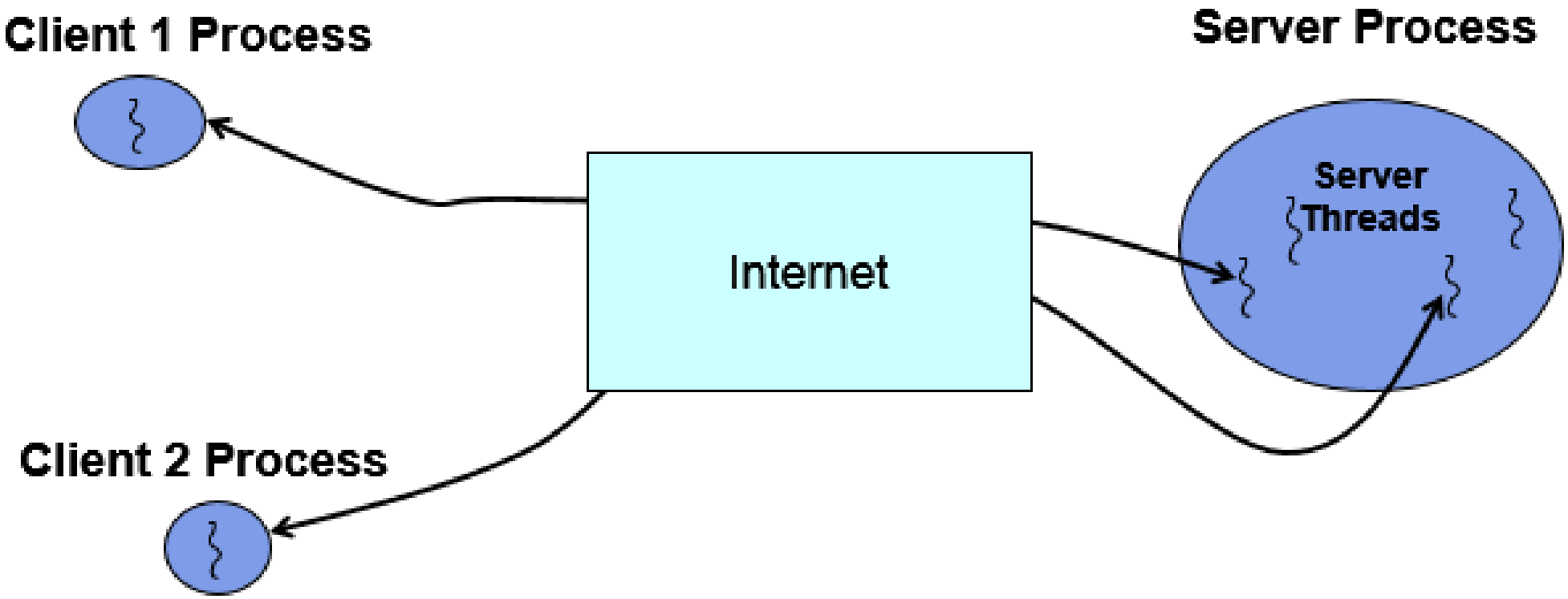
- We require that our computer can do multiple jobs at one time.
- Multiple processes & multiple threads.
- A thread of execution is a fork of a computer program into two or more concurrently running tasks.

Why do we need multi-threading?





Why do we need multi-threading?



Multi-threading in Java: Example

- To implement multi-threading in Java:
  - Implement *Runnable* interface
  - Extend from *java.lang.Thread* Class
- Today, we will implement it by extending from Thread class:
  - First, derive a class from *java.lang.Thread*
  - Then, override the *public void run()* method

```
public class ThreadWorker extends Thread {
        public void run() {
            // do your work here
    }
}
// create and start new thread
ThreadWorker w = new ThreadWorker()
w.start()
```

More on HTTP/TCP Programming

- A simple web server
- Multi-thread programming
- Web server with multi-threading

Web Server with multi-threading: WebServer_mt

```
package lab3;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import lab3.ThreadWorker;

public class WebServer_mt {
	public static void main(String[] args) throws IOException {
		// create server socket
		int nPort = 8090;
		ServerSocket svrSocket = new ServerSocket(nPort); // run the server;

		// keep listening on port 8090, serve for each req
		Socket clientSocket = null;
```

Web Server with multi-threading: WebServer_mt

```
while (true) {
                    System.out.println("Server is ready for new requst...");

                    // block until TCP connection is established
                    clientSocket = svrSocket.accept();

                    System.out.println("Server get a requst, assign to worker...");
                    // handle the rest by thread worker
                    ThreadWorker worker = new ThreadWorker(clientSocket);
                    worker.start();

            }
    }
```

Web Server with multi-threading: ThreadWorker

```
package lab3;
import java.net.*;
import java.io.*;
import java.lang.Thread;

public class ThreadWorker extends Thread {
        // client socket to handle
        private Socket clientSocket;
        // get client socket to handle when creating instance
        public ThreadWorker(Socket soc) {clientSocket = soc;}
        // override the thread method
        public void run() {
                long nCurrentWorkerID = Thread.currentThread().getId();
                System.out.println("worker " + nCurrentWorkerID + ": handling req...");
```

Web Server with multi-threading: ThreadWorker

```
try {
        // Get the HTTP request content
        BufferedReader inFromClient;
        inFromClient = new BufferedReader(new InputStreamReader( clientSocket.getInputStream()));
        char reqBuf[] = new char[1024];
        inFromClient.read(reqBuf);
        String strReq = new String(reqBuf);
        strReq.trim();
        System.out.println("work " + nCurrentWorkerID + ": req content = " + strReq);

        // what should we do here? parse req!
        String[] arrHttpContent = strReq.split(" ");
        String strMethod = arrHttpContent[0];
        String strTarget = arrHttpContent[1];

        if (strTarget.equals("/favicon.ico")) {
                System.out.println("browser is requesting icon, forget it!");
                return;// do nothing
        }
```

Why do we need this part?

Web Server with multi-threading: ThreadWorker

```
// send resp to client
String content = "<html><head><title>test</title></head><body>welcome to my homepage, <a
href='lab3.pdf'>pdf</a></body></html>";
String html = "HTTP/1.1 200 OK\n Connection:close\n Date: Mon, 23 Feb 2009 14:23:00 GMT\n"
        + "Server:Apache/1.3.0 (unix)\n Content-Length:“
        + content.length() + "\n"
        + "Content-Type: text/html\n\n“ + content;
DataOutputStream outToServer = new DataOutputStream(
                clientSocket.getOutputStream());
outToServer.writeBytes(html); // send out html
outToServer.flush();
clientSocket.close(); // close the socket;
System.out.println("worker " + Thread.currentThread().getId() + ": resp is sent.");
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace(); } }  }
```

Web Server with Multi-threading: Practice

- Compile and run this code
- Use web browser to test your server
- Carefully read the output, think about:
  a) Are sure the multiple threading is working as expected?
  b) How many HTTP requests are generated?
  c) What is the difference btw them?
  d) How can your server respond these req?

Code: https://www.dropbox.com/sh/p3ul8o30jmo4i1m/AAArF9KsBB0bQ99DBvvZ_Erta?dl=0

Q&A