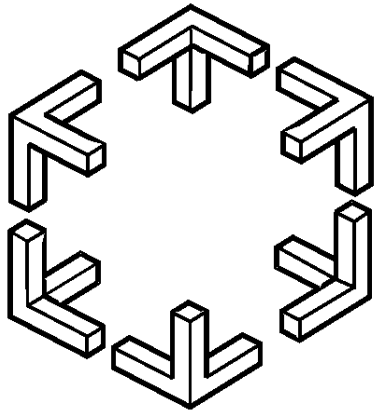
Distributed Computing and Systems

Chalmers university of technology



Multi-threaded Web Server (Assignment 1)

Georgios Georgiadis

Overview

- Multi-threaded Web Server
  - What to do and how to do it
  - HTTP messages
  - Processes and threads

Multi-threaded Web Server

- Motivation
  - Real application
  - Many applications use a web-based interface
    - e.g. service configuration, routers and firewalls

Multi-threaded Web Server

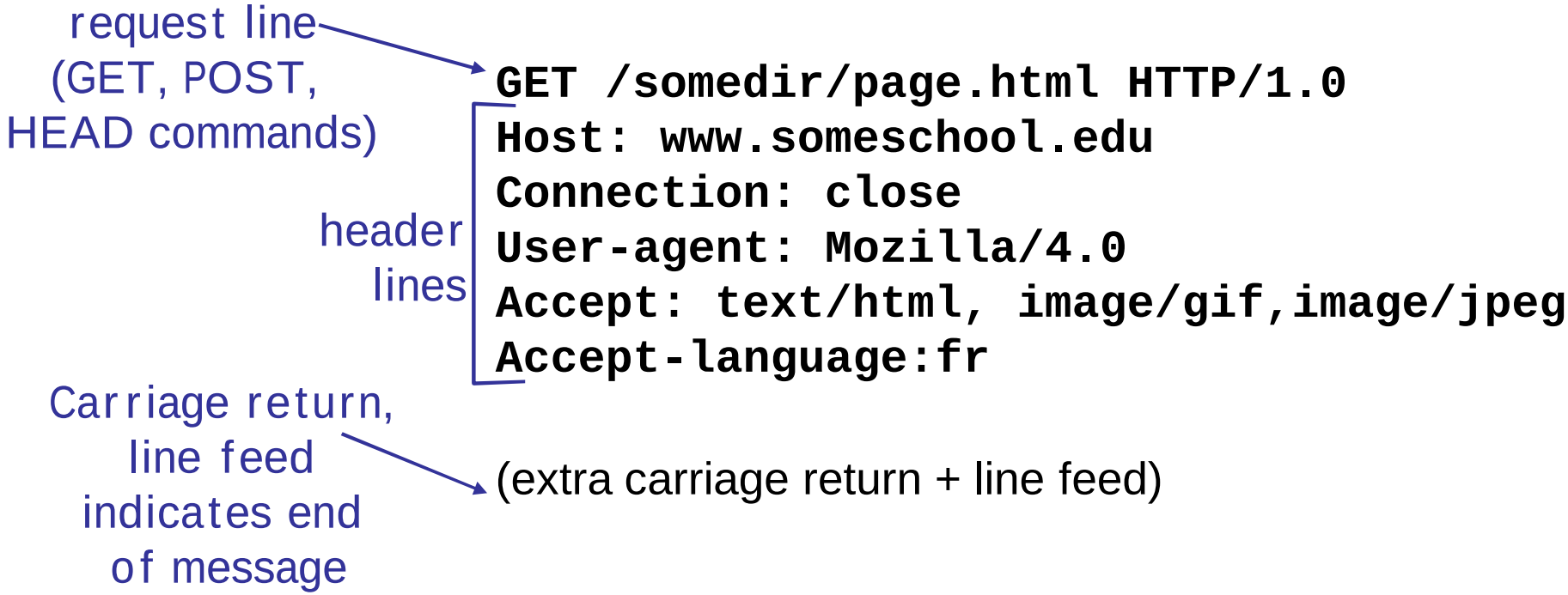
- The task:
  - Write a small Web server that supports a subset of the HTTP 1.0 specifications
  - The server should
    - be able to handle simultaneous requests
    - implement the HTTP methods GET and HEAD
    - handle and respond to invalid requests

Multi-threaded Web Server

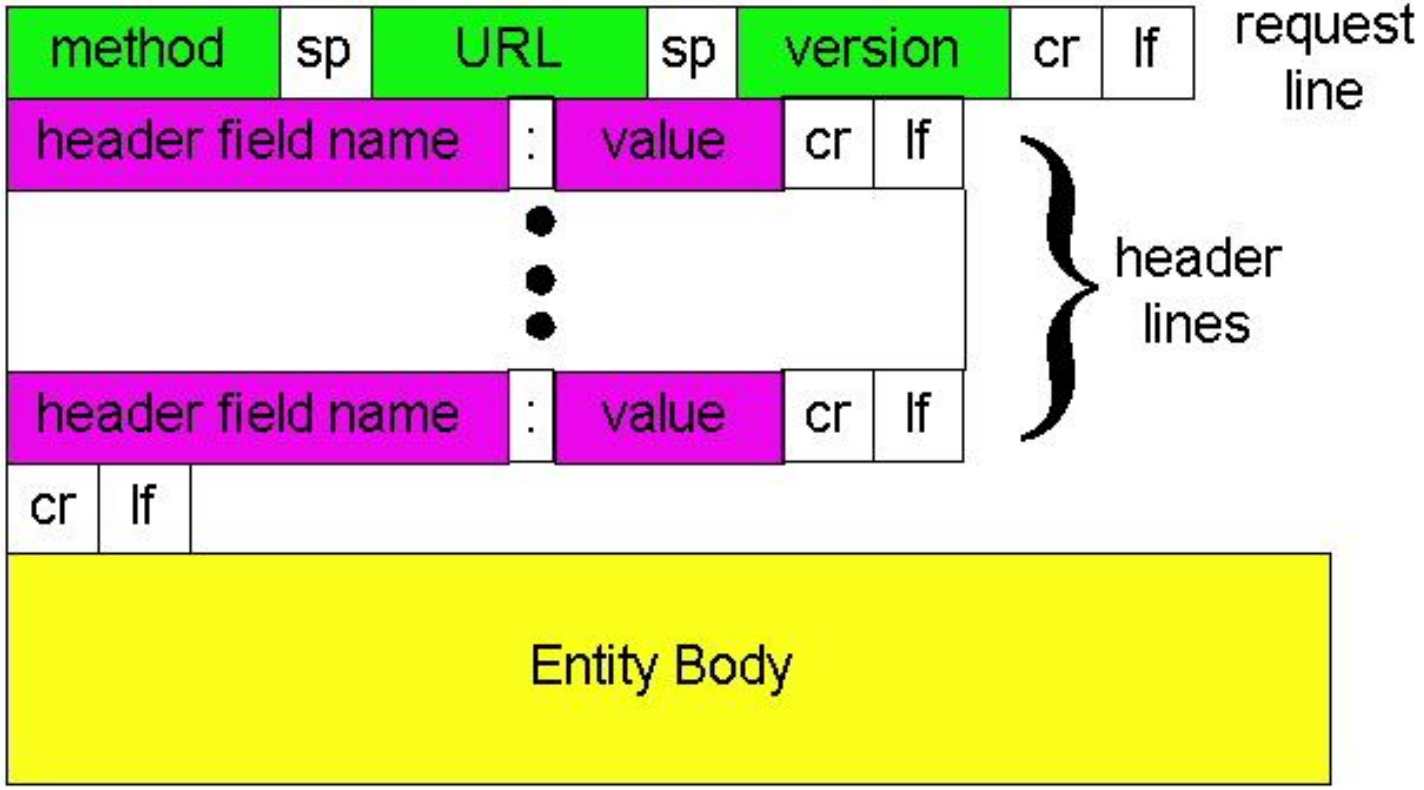
- Hints
  - Read the textbook
    - an example: simple Web server that does not handle simultaneous requests (Section 2.7, 2.9)
  - To handle concurrent requests
    - One way is to create a thread for each request
      - Java tutorial *Writing a Client/Server pair*
  - Check course webpage
    - Both FAQ and instructions

http message format: request

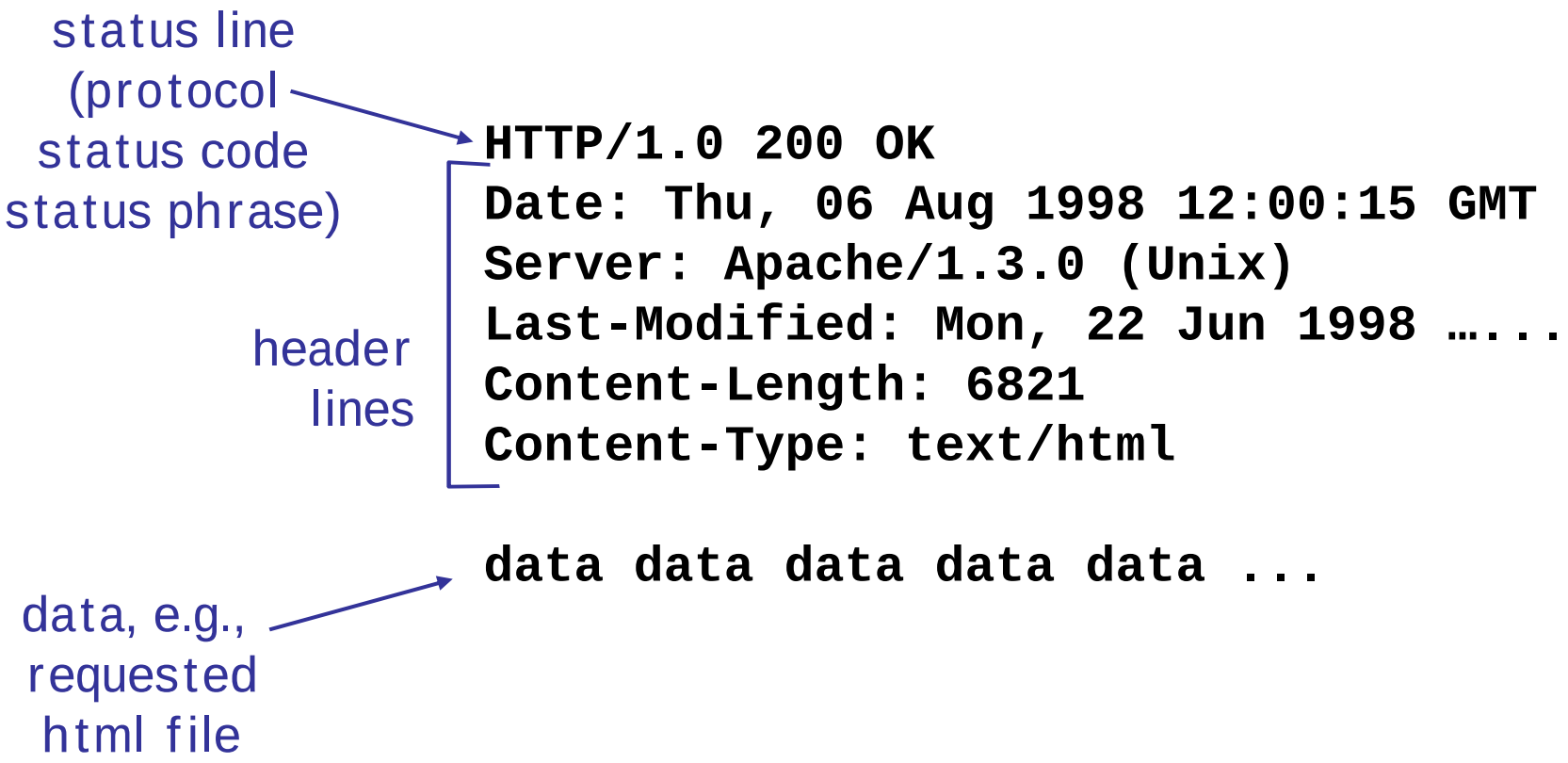
ASCII (human-readable format;
try telnet to www server, port 80)



http request message: general format



http message format: response



http response status codes

In first line in server->client response message.

A few sample codes:

200 OK

- request succeeded, requested object later in this message

301 Moved Permanently

- requested object moved, new location specified later in this message (Location:)

400 Bad Request

- request message not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

Processes and Scheduling

- The unit of execution (OS level)
- OS runs a collection of processes.
- Creates processes from programs
- Determines which process to run
- OS switches among processes
  - *context switch:* may cost a lot of CPU time.
  - occur at the end of each time slice.
  - each time a process invokes a system call

Process vs. Thread

- *Process*: an active entity which maintains its own set of resources
- *A thread* uses resources of its enclosing process
  - concurrent JVM composed of multiple threads
  - each thread acts like a single sequential JVM,
  - but shares access to a set of “passive” objects
  - Increased responsiveness to the user
  - maximum parallelization

Java Concurrency Support

```
class MessagePrinter implements Runnable {
 protected String msg_; The message to print
 protected PrintStream out_; The place to print it

 MessagePrinter(PrintStream out,String msg) {
   out_ = out;
   msg_ = msg;
 }
 public void run() {
   out_.print(msg_); // display the message
 }
}
```

Sequential Version

```
class SequentialPrinter {

  public static void main(String[] args) {

    MessagePrinter mpHello = new
  MessagePrinter("Hello\n", System.out);
    MessagePrinter mpGoodbye = new
  MessagePrinter("Goodbye\n", System.out);

    mpHello.run();
    mpGoodbye.run();
  }
}
```

MultiThreaded Version

```
class ConcurrentPrinter {

   public static void main(String[] args) {

     MessagePrinter mpHello = new
   MessagePrinter("Hello\n", System.out);
     MessagePrinter mpGoodbye = new
   MessagePrinter("Goodbye\n", System.out);
     Thread tHello = new Thread(mpHello);
     Thread tGoodbye = new Thread(mpGoodbye);
     tHello.start();
     tGoodbye.start();
   }
}
```

Different types of servers

- Single process/thread

```
do forever
    accept client connection
    process all client requests
    close connection
```

- One thread per connection

```
do forever
    accept client connection
    create a new thread to process requests
```

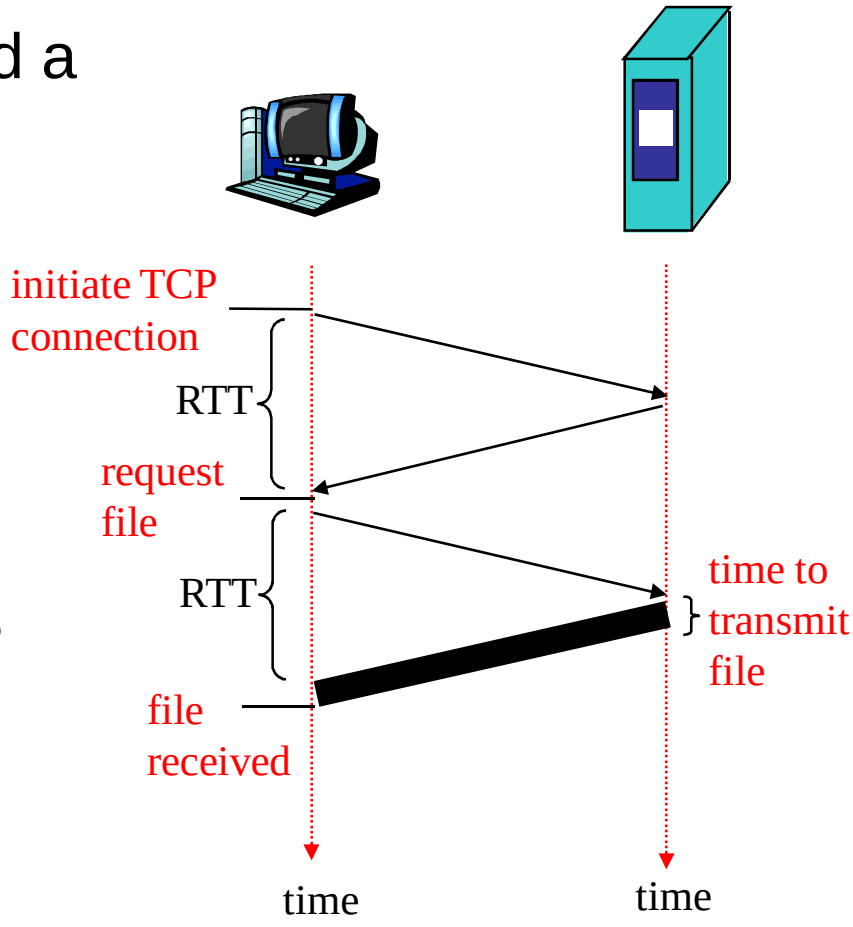
Response time

Definition of RTT: time to send a small packet to travel from client to server and back.

Response time:

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time

total = 2RTT+transmit time



How to avoid “Broken pipe”

- Server should run in the lab
- Client should run elsewhere
  - i.e. *remote1.studat.chalmers.se*
  - Also *remote{2,3,4,5}.studat* …
  - Use your CID credentials
  - Usage: WebServerTest [-p <port>] [-h <host>]

Skeleton code

<see pdf>

```java
//
// Multithreaded Java WebServer
// (C) 2001 Anders Gidenstam
// (based on a lab in Computer Networking: ..)
//

import java.io.*;
import java.net.*;
import java.util.*;
import java.net.InetAddress.*;

public final class WebServer
{
    public static void main(String argv[]) throws Exception
    {
    // Set port number
    int port = 0;

    // Establish the listening socket
    ServerSocket serverSocket = new ServerSocket(port);
    System.out.println("Port number is: "+serverSocket.getLocalPort());


    // Wait for and process HTTP service requests
    while (true) {
        // Wait for TCP connection
        Socket requestSocket = serverSocket.accept();

        // Create an object to handle the request
        HttpRequest request  = new HttpRequest(requestSocket);

        //request.run()

        // Create a new thread for the request
        Thread thread = new Thread(request);

        // Start the thread
        thread.start();
    }
  }
}

final class HttpRequest implements Runnable
{
    // Constants
    //   Recognized HTTP methods
    final static class HTTP_METHOD
    {
    final static String GET  = "GET";
    final static String HEAD = "HEAD";
    final static String POST = "POST";
    }

    final static String HTTPVERSION = "HTTP/1.0";
    final static String CRLF = "\r\n";
    Socket socket;

    // Constructor
    public HttpRequest(Socket socket) throws Exception
    {
    this.socket = socket;
    }

    // Implements the run() method of the Runnable interface
    public void run()
    {
```

```
67          try {
68              processRequest();
69          } catch (Exception e) {
70              System.out.println(e);
71          }
72          }
73
74          // Process a HTTP request
75          private void processRequest() throws Exception
76          {
77              // Get the input and output streams of the socket.
78          InputStream ins       = socket.getInputStream();
79          DataOutputStream outs = new DataOutputStream(socket.getOutputStream());
80
81          // Set up input stream filters
82          BufferedReader br = new BufferedReader(new InputStreamReader(ins));
83
84
85          // Get the request line of the HTTP request
86          String requestLine = br.readLine();
87
88          // Display the request line
89          System.out.println();
90          System.out.println("Request:");
91          System.out.println("  " + requestLine);
92
93          // Close streams and sockets
94          outs.close();
95          br.close();
96          socket.close();
97          }
98
99          private static void sendBytes(FileInputStream  fins,
100                        OutputStream     outs) throws Exception
101         {
102         // Coopy buffer
103         byte[] buffer = new byte[1024];
104         int    bytes = 0;
105
106         while ((bytes = fins.read(buffer)) != -1) {
107             outs.write(buffer, 0, bytes);
108         }
109         }
110
111         private static String contentType(String fileName)
112         {
113         if (fileName.toLowerCase().endsWith(".htm") ||
114             fileName.toLowerCase().endsWith(".html")) {
115             return "text/html";
116         } else if (fileName.toLowerCase().endsWith(".gif")) {
117             return "image/gif";
118         } else if (fileName.toLowerCase().endsWith(".jpg")) {
119             return "image/jpeg";
120         } else {
121             return "application/octet-stream";
122         }
123         }
124     }
125
```