Search Engines

WS 2009 / 2010

Lecture 2, Thursday October 29th, 2009

(Socket Communication, TCP/IP, HTTP, etc.)

Prof. Dr. Hannah Bast
Chair of Algorithms and Data Structures
Department of Computer Science
University of Freiburg

Rules for the Exercises

- Exercises are the most important part of this course
  - you may skip the lecture if you feel you don’t need it
  - you may skip the tutorials if you feel you don’t need it
  - but you absolutely must to the exercises
- You can’t work in groups
  - must do everything by yourself, otherwise you don’t learn it
  - if you cheat / copy, you are out, so don’t do it!
  - in the project after the lecture you can work in groups!
- Marks
  - one point per exercise, you will get a mark in the end
  - the exercise mark is 40% of your final mark, that’s a lot

The code you write …

- … should satisfy certain standards
  - at least minimally documented
    - a description at the beginning what the program does
    - a description of every class and every function
  - following some style guidelines, and do it consistently
    - see NoNos on next slide
  - think about naming of variables, classes, etc.
  - your code should always come with a README file that says
    - exactly how you compiled your program
    - exactly how you ran your program
    - describe any additional tools that you used

you will get less points if you don't care about this

Coding NoNos (a selection)

- Inconsistent spacing

```
if (flag   ==true){   x = x  +2   ;   flag=  false;}
```

- Inconsistent indentation
  - on same level always use, say, 2 spaces (never use tabs!)
  - place your { … } consistently
- Meaningless or incomprehensible names

```
class MyClass;
int stack = 3;
char* mstrfgy_W;
```

- Overlong methods
  - not more than, say, one screen per method

Oh yes, and for the other write-up ...

- … please also maintain a certain standard
  - proof-read before you submit
  - running a spell-checker is an absolute must
    - make it a habit!
  - whenever you do something you have to argue …
    - … how you have done it
    - … and why you did it the way you did it
    - e.g., you can’t just write: my ε is 0.06
  - the exercises are deliberately somewhat underspecified
    - whenever something is unclear, ask!

Goals for Lecture 2

- Search with a client and a server
  - in Lecture 1 / Exercise Sheet 1, you have learned how to build a (very simple) standalone search engine
  - in Lecture 2 / Exercise Sheet 2, learn how to build a browser-based search engine
    - client, server, and communication between the two
- Network communication
  - an important ingredient of every search engine
  - learn what is involved
  - and what makes it fast / slow

Overview of Lecture 2

- Socket Communication
  - basic principles
  - basic code
- TCP / IP
  - what is involved
  - how fast / slow
- HTTP
  - basic protocol
  - request types: GET, POST, etc.
- HTML
  - basic principle
  - forms, input, submit

Socket Communication

- First, some terminology
  - Process: program with its own resources (i.p. memory) running on your computer
  - How do processes communicate with each other?
  - Socket: communication point, like one end of a telephone line.
  - For us here Socket = IP address + Port.
  - IP address: the telephone number of your computer
  - Port: like a sub-telephone number
- Communication is two-way
  - both ends need a Socket = IP address + host

    (both sockets may be on the same computer though, e.g. for local inter-process communication)

Socket communication — Server Code

- Here is how server code looks like in C++   (simplified!)

```
server_fd = socket(AF_INET, SOCK_STREAM, 0)
server_address.sin_family = AF_INET;
server_address.sin_addr.s_addr = INADDR_ANY;
server_address.sin_port = htons(80);
bind(server_fd, &server_address);
listen(server_fd, 5)

client_fd = accept(server_fd, &client_address);
read(client_fd, buffer, 1024);
printf(“Here is the request I got: %s\n”, buffer);
write(client_fd, “Never say that again to me!”, 27);

close(client_fd);
```

many details ommitted, e.g., you must read and write in rounds!

Socket communication — Client Code

- Here is how client code looks like in C++ (simplified!)

```
client_fd = socket(AF_INET, SOCK_STREAM, 0);
server = gethostbyname(“vulcano.informatik.uni-freiburg.de”);
server_address.sin_family = AF_INET;
server_address.sin_addr.s_addr = server->h_addr;  // use bcopy
server_address.sin_port = htons(80);

connect(client_fd, &server_address);
write(client_fd, “Why me?”, 7);
read(client_fd, buffer, 1024);
printf(“Here is what the oracle told me: %s\n”, buffer);

close(client_fd);
```

for details refer to man pages or documentation on the web

Protocol, HTTP

- Processes need to agree on a protocol for the communication, e.g.
  - Process 1: How much is [mathematical expression]
  - Process 2: [mathematical expression] is [result]
- HTTP is a *very* simple protocol
  - Process 1: GET /index.html HTTP/1.1
  - Process 2:

```
HTTP/1.1 200 OK
Date: Thu, 29 Oct 2009 16:34:12 GMT
[empty line]
Here comes the answer to the request /index.html
```

More about HTTP

- HTTP can do more stuff though

  HEAD: just like GET, but only ask for the headers

  POST: send data along with the request

  (Note: small data can also be appended to URL in GET)

  PUT: Upload data to given URL (similar to FTP)

  DELETE: Delete that data

  TRACE: echo back request (with changes that happened underway)

  OPTIONS: ask which HTTP methods are supported

  CONNECT: convert request connection to tunnel

as a minimum GET and HEAD must be supported

Browser ↔ Webserver Communication

- What happens when you type a URL
  - say http://ad.informatik.uni-freiburg.de/teaching
  - browser creates an internet socket, as described
  - binds it to some free local port of your machine, e.g. 17457
  - get IP address for ad.informatik.uni-freiburg.de
    - for this browser has to ask a (nearby) DNS server
  - send HTTP request string to that machine on port 80

    GET /teaching HTTP/1.1 (and some optional headers)
  - receive answer with HTTP headers + newline + contents
    - one of the HTTP headers says that it is an HTML page

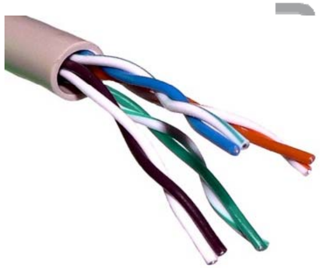
    Content-Type: text/html; charset=utf-8
  - browser renders the HTML in a nice way

TCP / IP

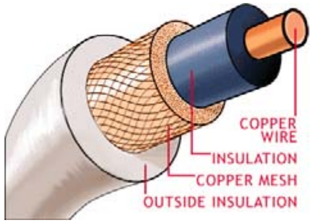
- Internet Protocol Suite (TCP / IP is the shortcut)
  - Link Layer e.g. Ethernet or WLAN
    - send packets along local links
  - Internet Layer e.g. IPv4 or IPv6
    - send packets across the Internet, unreliable
  - Transport Layer e.g. TCP or UDP
    - send packets across the Internet, reliably
  - Application Layer e.g. HTTP
    - send a request string, get an answer string
- And below all that is the hardware
  - twisted pair cables, coaxial cables, optical fiber

Hardware

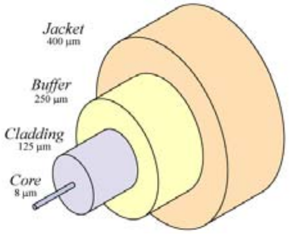
- Twisted Pair Cables
  - cheap, for distances up to 100m
  - bandwidth: 1 GBit / second



- Coaxial cables
  - more expensive, for distances up to 1000m
  - bandwidth: 10 GBit / second



- Optical fibre
  - much more expensive, great for long-distance
  - around 100 GBit / second per channel (frequency)
  - around 100 channels / fibre
  - around 100 fibres / cable



recall: typical disk transfer rate is 50 MB = 400 MBit / second

Link Layer — send packets along single link

- For example, Ethernet
  - Computers locally connected via cable (typically twisted pair Ethernet)
  - CSMA / CD protocol
    - CSMA = carrier sense multiple access
    - CD = collision detection
    - think of several people at a dinner table, only one person should speak at a time.
  - like this, send so-called frames of data
    - send bit after bit, abort if collision occurs
- Typical data transfer rate: 1 Gbit / second

Internet Layer — send across Internet, unrel.

- For example, IP = Internet protocol
  - send a packet of data from one computer to another
  - use Link Layer protocols for each link
  - packets consist of: source address, target address, data
  - routing is local: each router sends to locally next best router, based on prefix of target address
  - IP is unreliable:
    - packets may get lost
    - packets may get duplicated
    - packets may get distorted
    - packets may arrive out of order
- Typical data transfer rate: Exercise 4

Transportation Layer — TCP (reliable)

- TCP = Transmission Control Protocol
  - send packets reliably:
    - no packet loss or corruption, no out of order arrival
  - realized as follows:
    - connection establishment via three-way handshake
      - client SYN, server SYN-ACK, client ACK
    - data transfer via packet numbers and acks
      - destination host rearranges packets acc. to number
      - resent packages receipt of which was not ack’ed
      - discard duplicate packets
      - flow control (destination host has limited buffer)
      - congestion control (“slow start”, etc.)
- Typical data transfer rate: Exercise 4

Transportation Layer — UDP (unreliable)

- UDP = User Datagram Protocol
  - send messages via an unreliable Internet Layer protocol
    - messages may arrive out of order
    - messages can get lost
    - messages can get corrupted
  - thereby faster than TCP   how much: Exercise 2.3
  - unreliability is acceptable in many applications
    - DNS serving
    - video streaming, voice over IP, etc.
    - online games
- Typical data transfer rate: Exercise 4

Application Layer

- Send and receive following a certain protocol
- For example, HTTP
  - send a request string in a particular format
    - e.g. GET /xyz HTTP 1.1
  - receive an answer string in a particular format
    - HTTP headers + empty line + contents
  - all kinds of other fancy stuff
    - caching, keep connection open, etc.
  - reliability issues are handled by the underlying layer
    - typically TCP
- Typical data transfer rate: Exercise 4

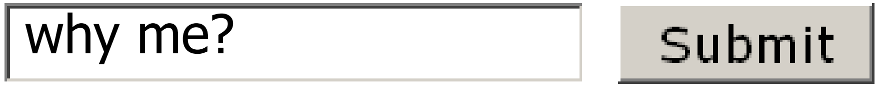
Finally, some HTML

- HTML = hypertext markup language
  - primary goal: basic markup for dummies
  - mixture between more semantic and purely layout markup

<h1> … </h1>	level-1 heading
<br />	line break

  - also contains communication semantics …
- Forms

```
<form action="http://some_url" method="GET">
  <input type="text" name="query" />
  <input type="submit" value="Submit" />
</form>
```


why me?

  - will send GET request to http://some_url/?query=why+me%3f