

EECS 22L: Software Engineering Project in C Language

Lecture 9

Rainer Dömer

doemer@uci.edu

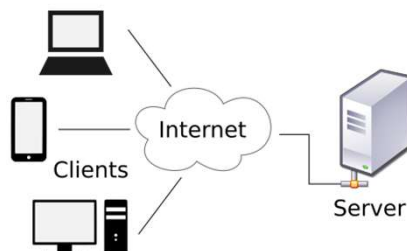
The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 9: Overview

- Project 2 Technical Discussion and Advise
 - Client-server software architecture
- Discussion on Socket Communication
 - Client-server example
 - Blocking I/O communication
 - Multiplexing multiple connections
 - Clock server example

Project 2: Software Architecture

- Client-Server Software Architecture
 - Software applications communicating via the Internet



Source: David Vignoni (LGPL, wikipedia.org)

- Server: provides a service function to one or more clients
- Client: initiates requests for service
- Internet: communication network to exchange messages

Project 2: Client-Server Communication

- Discussion on Socket Communication
 - Sockets Tutorial
 - http://www.linuxhowtos.org/C_C++/socket.htm
 - <http://www.linuxhowtos.org/data/6/client.c>
 - <http://www.linuxhowtos.org/data/6/server.c>
 - Reference: Linux manual pages
 - `man socket`
 - `man select`
 - `man select_tut`
 - Extended client-server example:
 - `~eecs22/SocketTutorial.tar.gz`
 - `client2.c`
 - `server2.c`
 - This example can handle only one client at a time, others have to wait for their turn to connect

Project 2: Client-Server Communication

- Discussion on Socket Communication
 - Sequence Diagram for client-server example

```

sequenceDiagram
    participant Client
    participant Server
    Client->>Client: socket()
    Server->>Server: socket()
    Server->>Server: bind()
    Server->>Server: listen()
    Client->>Server: connect()
    Server->>Server: accept()
    Client->>Server: write()
    Server->>Server: read()
    Server->>Client: read()
    Client->>Server: read()
    Server->>Server: write()
    Client->>Server: close()
    Server->>Server: close()
    Client->>Client: exit()
    Server->>Server: exit()
    
```

- This simple example can handle only one client at a time, others have to wait for their turn to connect (they are blocked)
- *Blocking communication* can stall both the client and the server!

EECS22L: Software Engineering Project in C, Lecture 9 (c) 2018 R. Doemer 5

Project 2: Client-Server Architecture

- Communication Example: *Initial Chat App*
 - Client: `Hello!`
 - Server: `ERROR invalid message "Hello!"`

 - Client: `CREATE_ACCOUNT Albert Einstein AE`
 - Server: `OK AE = Albert Einstein`

 - Client: `GET_CONTACTS AE`
 - Server: `OK IN = Isaac Newton, NT = Nikola Tesla`

 - Client: `SEND_MESSAGE NT Hello Nikola, how are you?`
 - Server: `OK`

 - Client: `RECEIVE_MESSAGES`
 - Server: `OK NT="I'm fine, thanks!", END_OF_MESSAGES`

Avoid a possibly long delay for the other client to respond!

EECS22L: Software Engineering Project in C, Lecture 9 (c) 2018 R. Doemer 6

Project 2: Client-Server Architecture

- Discussion on Socket Communication
 - Handling multiple active client connections
 - Option 1: Parallel/concurrent (asynchronous) I/O
 - Use multiple processes (`fork()`) or threads (`pthread_create()`)
 - Requires Operating System (OS) knowledge (i.e. EECS 111), and very careful programming to avoid race-conditions and deadlocks
 - Option 2: Synchronous I/O multiplexing
 - Wait on multiple I/O requests, handle them first-come-first-served (FCFS)
 - Function `select()` monitors multiple file descriptors, with timeout option
 - Multiplexing multiple connections with `select()`
 - Clock server example: `~eecs22/ClockServer.tar.gz`
 - `ClockServer.c`
 - `ClockClient.c`
 - `Makefile`, `README`
 - Online demonstration!

EECS22L: Software Engineering Project in C, Lecture 9 (c) 2018 R. Doemer 7

Project 2: Client-Server Architecture

- Multiplexing multiple client connections with `select()`
 - ClockServer example: `~eecs22/ClockServer.tar.gz`

```

sequenceDiagram
    participant CS as ClockServer
    participant CC as ClockClient
    CS->>CS: socket()
    CS->>CS: bind()
    CS->>CS: listen()
    CS->>CS: select()
    CC->>CC: socket()
    CC->>CC: connect()
    CC-->>CS: connect
    CS->>CS: accept()
    CS->>CS: read()
    CS-->>CC: data
    CC->>CC: read()
    CC->>CS: write()
    CS->>CS: write()
    CS->>CS: printf()
    CS-->>CS: time-out
    CS->>CS: fflush()
    CS->>CS: close()
    CS->>CS: exit()
    CC->>CC: close()
    CC->>CC: exit()
    
```

- Wait simultaneously to connect, to transfer data, or for time-out!
- Keep sequential execution short
- Limit client-server interaction to one request at a time

EECS22L: Software Engineering Project in C, Lecture 9 (c) 2018 R. Doemer 8