et2440: "TCP/IP"

Laboratory exercise: **Interprocess Communication**

Blekinge Institute of Technology

School of Engineering

Department of Telecommunication Systems

Name: _____ Acronym: _____

Examiner: _____ Date: _____ Pass: _____

Contents

1 Introduction

1.1 Goals and ambitions

After having completed the following exercises you will:

- Know how to write TCP and UDP client and server programs.
- Have implemented five simple client-server applications.

1.2 Recommended reading

All the information you need to complete the exercises is available online. We particularly recommend Beej's Guide to Network Programming (`http://beej.us/guide/bgnet/`).

1.3 Programming Languages and Preliminaries

You will benefit from rudimentary skills in navigating a UNIX system, launching programs and listing directories. Being able to use either emacs or vi is a definite advantage.

You may choose to implement using one of the following languages: C, C++ or Java. No other programming languages are permitted.

1.4 Formatting

The text uses different typefaces to distinguish between user input and the output of commands. The italic Courier typeface signifies user input, whereas regular Courier typeface denotes the output from a command. It

is also used for source code listings.

Example

Issuing the command:

ls -l valid.xml

will yield the following results:

```
-rw-r--r-- 1 der dave 223 Mar 14 14:13 file.c
```

The file looks like:

file.c

```
1
2  /* @(#)file.c
3   */
4
5  #include <config.h>
6  #include "file.h"
7
8  void main(void) {
9      printf("hello world\n");
10 }
11
```

1.4.1 System Environment

This lab exercise will be examined on either a UNIX (Linux, typically) or using the Cygwin environment for Windows.

1.5 Deliverables

The following items must be provided by you to pass the exercise:

1. A short document giving answers to any questions posed in the exercises. The only acceptable formats are pure text or PDF. UTF-8 or pure ASCII text is preferred. In particular, Word documents are not permissible! The document should be named '`acronym_report.txt`' or '`acronym_report.pdf`', where you should replace 'acronym' with your own acronym, e.g., `lier01_report.txt`
2. All source code and other documents created during the exercises. These should be properly named as indicated in this text.
3. Compilation and execution instructions. This file should be called `README` or `README.txt`.

Delivery of the files should be in a zip or tar.gz file, in which the files should be placed in a sub-directory called `acronym_et2440_lab_ipc`. The file itself should be called `acronym_et2440_lab_ipc.zip` or `acronym_et2440_lab_ipc.tar.gz`. Again, you should replace 'acronym' with your own acroynm, e.g., `lier_et2440_lab_ipc.tar.gz`.

If the archive is corrupt (i.e., if it is not possible to extract all files), if the correct name format is not used, or the archive does not contain a directory, you will fail examination.

Information on delivery dates will be disseminated separately.

1.6 Examination

During the lab examination, make sure your software is loaded and configured on your student account or your own laptop before presenting the lab. There is only a certain amount of time alotted for each examination, and the lab examiner will not wait for those who must copy/configure the software from removable media. Make sure your software is running without any problems.

Lab assistance prior to the examination will be provided via the forum on the BTH LMS, IT's Learning.

1.7 Code of Ethics

The exercises are to be performed, written and implemented *individually*.

Sometimes it is necessary to ask an advisor or friend for help. This is permitted and encouraged, provided that:

- You give due credit in the relevant files by placing a comment block at the top of the file.
- You fully understand the part which you did not solve yourself.

Any other form of cooperation, copying and/or sharing of solutions will be considered a breach of the code of ethics and will warrant loss of credit and/or additional exercises. Specifically, trying to submit someone else's work as your own warrants reporting of the incident to the disciplinary committee.

2 Exercises

2.1 Preliminary exercises

Before starting the labs, you'll need to find out some information about the computer you are working on, and answer a few questions regarding the topic of IPC.

2.1.1 Host information

For each of the Ethernet interfaces (wired and wireless), fill in the following table.

2.1.2 Questions

1. Briefly explain the term IPC in terms of TCP/IP communication.
2. What is the maximum size of a UDP datagram? What are the implications of using a packet-based protocol as opposed to a stream protocol for transfer of large files.
3. TCP is a reliable transport protocol, briefly explain what techniques are used to provide this reliability.
4. Why are the `htons()`, `htonl()`, `ntohs()`, `ntohl()` functions used?
5. What is the difference between a datagram socket and a stream socket? Which transport protocols do they correspond to?
6. What is a stateful service, as opposed to a stateless service? What are the performance implications of statefulness and statelessness?

2.2 Simple UDP and TCP Exercises

In this set of exercises, you will learn how to implement very simple services, that do not require any specific PDU interchange. Sessions are created by the fact of either connecting to a TCP or UDP service.

It is OK for the services below to function without DNS support.

Note: For all the protocols referenced here, the port numbers mentioned in the associated RFCs will most likely not work, so you may substitute them for any port number above 1024 of your choice.

2.2.1 Exercise 1 – Daytime protocol

Your task in this exercise is to implement the Daytime protocol, as defined in RFC867 (`http://www.faqs.org/rfcs/rfc867.html`).

You should write both a UDP and TCP server, as well as clients that access the services.

The deliverables should be called `daytimed_udp.c`, `daytimed_tcp.c`, `daytime_udp.c` and `daytime_tcp.c`, if you have chosen "C" as a language for the exercises. Otherwise, use the appropriate file extension.

2.2.2 Exercise 2 – Time protocol

Your task in this exercise is to implement the Time protocol, as defined in RFC868 (`http://www.faqs.org/rfcs/rfc868.html`).

You should write both a UDP and TCP server, as well as clients that access the services.

The client should display the result back to the user in a "nice" way. This means that you should not just dump the raw 32-bit data you get from the server on the command line, but present it in a humanly readable fashion.

The deliverables should be called `timed_udp.c`, `timed_tcp.c`, `time_udp.c` and `time_tcp.c`, if you have chosen "C" as a language for the exercises. Otherwise, use the appropriate file extension.

2.2.3 Exercise 3 – Echo protocol

Your task in this exercise is to implement the Echo protocol, as defined in RFC862 (`http://www.faqs.org/rfcs/rfc862.html`).

You should write both a UDP and TCP server, as well as clients that access the services.

The deliverables should be called `echo_udp.c`, `echod_tcp.c`, `echo_udp.c` and `echo_tcp.c`, if you have chosen "C" as a language for the exercises. Otherwise, use the appropriate file extension.

2.3 TCP and UDP Exercises

In this set of exercises you will learn how to create services that require more complex protocol interactions, as well as maintaining state in the server.

The services below should have DNS support, i.e., the client should be able to access the server via its hostname, not only its IP address.

For each protocol interaction, i.e., for each request-response pair, you should report the time the request took, and you should supply the approximate speed for each download and upload of a file.

Note: For all the protocols referenced here, the port numbers mentioned in the associated RFCs will most likely not work, so you may substitute them for any port number above 1024 of your choice.

2.3.1 Exercise 1 – TFTP

Your task in this exercise is to implement the Trivial File Transfer Protocol (TFTP), as defined in RFC1350 (`http://www.faqs.org/rfcs/rfc1350.html`).

You should write both a client and a server.

The deliverables should be called `tftpd.c` and `tftp.c`, if you have chosen "C" as a language for the exercises. Otherwise, use the appropriate file extension.

2.3.2 Exercise 2 – SFTP

Your task in this exercise is to implement the Simple File Transfer Protocol (SFTP), as defined in RFC913 (`http://www.faqs.org/rfcs/rfc913.html`).

You should write both a client and a server.

The deliverables should be called `sftpd.c` and `sftp.c`, if you have chosen "C" as a language for the exercises. Otherwise, use the appropriate file extension.

2.4 Hints and Tips

Debugging tools Before you start writing your clients, consider using `telnet` and/or `netcat` as debugging tools. Both programs allow you to have a generic client for testing the services. Telnet is for TCP connections only, and netcat allows you to test UDP services as well.