Design Document

Home Exam 2
Course: IN4320
Semester: Fall - 2019
Candidate nr: 15202

Questions

What problem occurs if two files are sent simultaneously from two different sources to the same receiver and port 30? Why? How can the problem be solved?

Answer:
If two files are sent from two different sources (MIP Transport daemons), the receiver (MIP Transport daemon) has no mechanism for separating the two streams of data from each other. The result would be that a file containing data from both sources is stored. For example, the problem could be solved by separating the two streams of data using the origin of the MIP address. Or to implement a session system.

How to solve the problem in the Internet, e.g. considering a web server listening on the same port (80) wherever connections come from?

Answer:
Web servers solve the problem of multiple clients by separating the clients into sessions controlled by the server. Sessions mean that the server and client are associated with an ID (session ID) so that both can keep track of who they are talking to at any time.

How can the timeout value be calculated automatically so that it is not needed as a command line argument?

Answer:
We could implement support for the MIP Transport daemon that sends to calculate the time it takes between the ACKs from the recipient. Using an average of the ACK times and a margin percentage (maybe twice the average time), the system could utilize a dynamic timeout.

Design

Building the program

Running a terminal in the project root

BUILD WITH DEBUG
make dev

BUILD ALL PROJECT EXECUTABLES
make

BUILD AND RUN TESTS
make tests

Executing the program

The executable files are located in the bin/ directory in the project root.
NOTE: all the daemons and applications require sudo to run

START THE MIP DAEMON
mipd [-h] [-d] <socket_application> <route_socket> <forward_socket> [MIP addresses ...]

START THE ROUTER DAEMON
The mipd process routerd is to connect to must have been started before attempting to start
routerd [-h] [-d] <route_socket> <forward_socket> [MIP addresses ...]

START THE PING CLIENT
ping_client [-h] <destination_host> <message> <socket_application>

START THE PING SERVER
ping_server [-h] <socket_application>

START THE MIP TP DAEMON
miptpd [-h] [-d] <mipd app socket> <app socket> <timeout>

NOT COMPLETED

START THE MIP TP CLIENT
tpclient [-h] [-d] <miptpd socket> <mip address> <port> <file>

START THE MIP TP SERVER
Not implemented

Project Layout

IN4230-HE
├── bin/ **<- Directory containing the compiled executables**
├── build/
├── commons/ **<- source code for common libraries and definitions**
├── mipd/ **<- MIP daemon source files**
├── ping-client/ **<- Ping client source files**
├── ping-server/ **<- Ping server source files**
├── routerd/ **<- Router daemon source files**
├── miptpd/ **<- Transport daemon source files**
├── tpclient/ **<- Transport client source files**
├── tpserver/ **<- Transport server source files**
├── LICENSE
├── Makefile **<- Full project Makefile, builds and tests all project artifacts**
└── README.md

Project structure for mipd and routerd

The daemons have a project structure with src and tests directories at the top and with lib and main directories in the src directory. The source code is split in library code and main code to facilitate unittesting of the daemon components through static linking.

```
miptpd/
├── build
├── src
│   ├── lib          <- Code tested by unit tests
│   └── main         <- Entrypoint code for running the program
└── tests            <- Unittests for the program
```

Disclaimer

I did not complete the implementation of the transport client and transport server. I simply ran out of time. The way the system is implemented the client and server connects to the tp daemon and either sends a blob of data to be transported (max 2^{16}) or signals the daemon with a flag that it is ready to receive. The transport daemon is responsible for streaming the data and provides back an acknowledgement to the sender when the job is done and a reassembled blob of data to the receiver. The transport server would be responsible for turning the blob of bytes received from the transport daemon into a file on the disk.

The Transport daemon implements Go-N-Back as the sliding window algorithm

Dependencies

The project uses some utility libraries for handy data structures such as lists and queues:

commons/src/hashmap.h/c
commons/src/list.h/c
commons/src/darray.h/c
commons/src/queue.h/c

https://github.com/zedshaw/liblcthw/tree/master/src/lcthw

Bstring library

commons/src/bstrlib.h/c

where gathered from: http://bstring.sourceforge.net/