Robin Lunde – robinlu – 27.02.2015 - INF3190

Difference between MIP and IPv4:

The biggest difference is the complexity. IPv4 can handle a few more errors and includes a check-sum, to make sure the header is intact. These things are lacking in MIP. The advantage of this is that MIP is a small protocol and as such very easy and fast to handle. It does however mean there is a larger chance of errors occurring, and fewer ways to handle such errors.

Graph of function calls:

MIP-Daemon for the client is called A.

MIP-Daemon for the server is called B.

1. Client calls write and sends message to A. B enters select and accepts connection on IPC-socket. Saves connection in Accpt-socket
2. A enters select and accepts connection on IPC-socket. Saves connection in Accpt-socket
3. A enters select and finds the Accpt connection. Calls recIPC() and receives message from client.
4. A calls decodeBuf() and separates the message and the destination MIP-address.
5. A calls findArp() and check if the destination MIP-address is in the ARP-cache
    1. **MIP is in ARP-cache:**
        1. A calls setTransport() to create a MIP-frame
        2. A calls createEtherFrame() and creates an Ethernet-frame
        3. A calls sendRaw() which sends the frame to the recipient.
        4. B enters select and finds connection on the raw socket.
        5. B calls recRaw() and receives the Ethernet-frame.
        6. B calls findCase() and decides what type of MIP-frame it has received by calling arpRet() and arp(). Detects a transport-frame.
        7. B calls sendIPC() and sends the message to the server.
        8. Server calls read() and receives the message.
        9. Server calls write() and sends “Pong” to B.
        10. B enters select and finds connection on Accpt socket.
        11. B calls decodeBuf() and tries to decode message.
        12. B calls findARP(). Since it was found in A, it has to exist in B (see MIP not in ARP-cache for explanation).
        13. B calls setTransport() to create MIP-frame
        14. B calls createEtherFrame() to create Ethernet-frame.
        15. B calls sendRaw() which send the frame to the recipient.
        16. A enters select and finds connection on raw socket.
        17. A calls recRaw() which receives the Ethernet-frame.
        18. A calls findCase() and decides what type of MIP-frame it has received by calling arpRet() and arp(). Detects a transport-frame.
        19. A calls sendIPC() and sends the message to the Client.
        20. Client calls read() and receives the message, prints it and how long it took to receive an answer (if it took under 1 second).
        21. Client closes.

2. **MIP is not in ARP-cache:**
    1. A calls setARP() and creates an ARP-MIP-frame.
    2. A calls createEthernetFrame() and creates and Ethernet-frame.
    3. A calls sendRaw() and sends the message.
    4. A calls setTempTrans() and saves as much of the information it can in a new MIP-frame for later. (This MIP-frame is hereby referenced as Temp MIP-frame).
    5. B enters select and finds connection on the raw socket.
    6. B calls recRaw() and receives the Ethernet-frame.
    7. B calls falls findCase() and decides what type of MIP-frame it has received by calling arpRet() and arp(). Detects ARP-MIP-frame.
    8. B calls findARP() and checks if the connection is saved in the ARP-cache. If it is not, it calls saveARP() and saves the connection in the ARP-cache.
    9. B calls setARPReturn() and creates an ARP-return-MIP-frame.
    10. B calls createEtherFrame() and creates an Ethernet-frame.
    11. B calls sendRaw() and sends the message.
    12. A enters select and finds connection on the raw socket.
    13. A calls recRaw() and receives the Ethernet-frame.
    14. A calls findCase() and decides what type of MIP-frame it has received by calling arpRet() and arp(). Detects ARP-return-frame.
    15. A calls saveArp() and saves the connection in the ARP-cache.
    16. A calls finalTransp() and finalizes Temp MIP-frame, with the last data.
    17. A calls createEtherFrame() and creates an Ethernet-frame.
    18. A calls sendRaw() and sends the message.
    19. B enters select and finds connection on the raw socket.
    20. B calls recRaw() and receives the Ethernet-frame.
    21. B calls findCase() and decides what type of MIP-frame it has received by calling arpRet() and arp(). Detects a transport-frame.
    22. B calls sendIPC() and sends the message to the server.
    23. Server calls read() and receives the message.
    24. Server calls write() and sends “Pong” to B.
    25. B enters select and finds connection on Accpt socket.
    26. B calls decodeBuf() and tries to decode message.
    27. B calls findARP(). Step 8 ensures that the connection is in the ARP-cache.
    28. B calls setTransport() to create MIP-frame
    29. B calls createEtherFrame() to create Ethernet-frame.
    30. B calls sendRaw() which send the frame to the recipient.
    31. A enters select and finds connection on raw socket.
    32. A calls recRaw() which receives the Ethernet-frame.
    33. A calls findCase() and decides what type of MIP-frame it has received by calling arpRet() and arp(). Detects a transport-frame.
    34. A calls sendIPC() and sends the message to the Client.
    35. Client calls read() and receives the message, prints it and how long it took to receive an answer (if it took under 1 second).
    36. Client closes.

Compiling and executing:

Compiling:

Compile regularly: make

Compile with debug option: make debug

Compile with extra error-detection: make more

Remove all compiled files: make clean

Remove compiled server-files: make cleanserver

Remove compiled client-files: make cleanclient

Remove compiled daemon-files: make cleandaemon

Execution:

Client: ./Client <MIP-daemon-name> <MIP-reciever-address> <message>
Server: ./Server <MIP-daemon-name>
Daemon: ./Daemon <MIP-daemon-name> <Interface*>
* Interface is the network-adapter you want the raw-socket to use for sending and receiving packages.

List of files:

Program:

Client.c

Server.c

Daemon.c

Protocol.c

Transfer.c

Structs.h

Other:

Makefile

DesignDocument.pdf

Notes:

The program reports memory-leaks in the daemon, but the leaks are always of 0 bytes. If I try to free the variables, I get a x is on top of thread 1's stack. I assume this is because it is used in the other daemon and makes no difference since there is no ACTUAL leak.

The amount of connections the daemon can accept, can be changed by the define maxCon in Daemon.c. It is currently set to 1000.

The program should be ran with initiating the daemons first, then initiating the server and lastly the client.

There can only be one client or server connected to a daemon.

If the program crashes, the file-descriptors may not be closed correctly and have to be deleted, if you want to try with the same file-descriptors again. If not an error will occur.