Exercise sheet 8 – Process communication 1

Goals:

- Understand signals
- Network socket programming (client/server)

Exercise 8.1: Signal handling

(a) Update the `OS_exercises` repository with `git pull`.

Proposal for solution: `git pull`

(b) Change into the `OS_exercises/sheet_08_process_comm1/signal` directory.

Proposal for solution: `cd OS_exercises/sheet_08_process_comm1/signal`

(c) Inspect the `signal_example.c` program.

(d) Run the `signal_example` program.

Proposal for solution: `./signal_example`

(e) Send a `SIGHUP` to the running `signal_example`. What do you expect? What happens?

Proposal for solution: `kill -SIGHUP pid`
The program prints that it received the SIGHUP signal.

(f) Send a `SIGINT` to the running `signal_example`. What do you expect? What happens?

Proposal for solution: `kill -SIGINT pid`
The program prints that it received the SIGINT signal.

(g) Send a `SIGQUIT` to the running `signal_example`. What do you expect? What happens?

Proposal for solution: `kill -SIGQUIT pid`
The program prints that it received the SIGQUIT signal.

(h) Send a `SIGTERM` to the running `signal_example`.

Proposal for solution: `kill pid`
The program prints that it received the SIGTERM signal.

(i) Send a `SIGKILL` to the running `signal_example`. Is `signal_example` still running? Is it possible to register to this signal inside the `signal_example.c`?.

Proposal for solution: `kill -SIGKILL pid`
The program has been killed. It's not possible to register the `SIGKILL` signal, sending this signal always kills the program.

(j) Run the `signal_example` program with the parameters `--abort`. What happens here?

Proposal for solution: The program gets instant the signal `SIGABRT` prints a message and quits.

(k) Run the `signal_example` program with the parameters `--alarm 10`. What happens here?

Proposal for solution: The program gets the signal `SIGALRM` after 10 seconds, then it prints a message and quits.

Exercise 8.2: Chat client/server: network sockets

(a) Change into the `sheet_08_process_comm1/nw_chatserver` directory.

Proposal for solution:

```
cd OS_exercises/sheet_08_process_comm1/nw_chatserver
```

(b) Inspect the `nw_chat_server.c`.

(c) Inspect the `nw_chat_client.c`.

(d) Complete `nw_chat_client.c`.

Proposal for solution:

```c
#include <stdio.h>      //printf
#include <stdlib.h>     //EXIT_SUCCESS, EXIT_FAILURE
#include <string.h>     //strcmp
#include <stdbool.h>    //true, false
#include <sys/socket.h> //socket, bind, listen, accept, recv, send
#include <netinet/in.h> //struct sockaddr_in
#include <unistd.h>     //close
#include <arpa/inet.h>  //inet_aton
#include <pthread.h>    //pthread_*

/*
 * nw_chat_client.c
 * The client for a simple chat server
 */

const int MAX_MESSAGE_LEN = 1024;  //Max length of messages
const int PORT            = 15000; //Network port

int network_socket = -1;

//this function receives all incoming messages, it should run inside a second thread
void* receiver_thread() {
    //endless loop to receive messages from the server
    while(true) {
        //receive data
        char received_message[MAX_MESSAGE_LEN];
        ssize_t size = recv(network_socket, &received_message, MAX_MESSAGE_LEN-1, 0);
        if(size <= 0) {
            break; //no data received or connection closed
        } else {
            //the message has to be properly 0-terminated
            received_message[size] = '\0';
            printf("Received: %s", received_message);
        }
    }
    return NULL;
```

```c
}

int main(int argc, char** argv) {
    //check if a parameter for the IP address exists
    char* server_ip = NULL;
    if(argc < 2) {
        printf("Usage: %s <serveraddress>\n", *argv);
        exit(EXIT_FAILURE);
    } else {
        server_ip = argv[1];
    }

    //create socket for outgoing connection
    network_socket = socket(AF_INET, SOCK_STREAM, 0);
    if(network_socket < 0){
        printf("Error: can't create socket!\n");
        exit(EXIT_FAILURE);
    }

    //connect to server
    struct sockaddr_in address;
    address.sin_family      = AF_INET;
    inet_aton(server_ip, &address.sin_addr); //convert internet host address to binary
    address.sin_port        = htons(PORT); //convert values between host and network b

    int connection_result =
        connect(network_socket, (struct sockaddr*) &address, (sizeof address));
    if(connection_result != 0) {
        printf("Error: can't connect to address: %s::%d\n", server_ip, PORT);
        exit(EXIT_FAILURE);
    }

    //start the thread to receive messages from the server
    pthread_t thread_id = -1;
    pthread_create(&thread_id, NULL, &receiver_thread, NULL);

    //send input from stdin as message
    char message[MAX_MESSAGE_LEN];
    while(true) {
        //fetch user input from console (stdin)
        fgets(message, MAX_MESSAGE_LEN, stdin);

        if(strcmp(message, "\\quit\n") == 0) {
            //close the network socket:
            // - similar to close(network_socket)
            // - but the recv() in the receiver_thread exits with: size == 0
            shutdown(network_socket, SHUT_RDWR);
            break;
        }

        //send message to the server
        send(network_socket, &message, strlen(message), 0);
    }

    //wait until the receive thread exits
    pthread_join(thread_id, NULL);

    //close socket
    close(network_socket);

    return EXIT_SUCCESS;
}
```

(e) Compile your program into `nw_chat_client`. Use the prepared `Makefile` with the target `nw_chat_client` for this!

Proposal for solution: `make`

(f) Start the provided `nw_chat_server` or use the `nw_chat_server` provided by the lecturer.

Proposal for solution: `./nw_chat_server`

(g) Start your chat client with `nw_chat_client <ip>` and chat. You may use a separate shell for that. You can exit your client by typing `\quit` and press enter.

Proposal for solution: `./nw_chat_client 127.0.0.1`