



ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ  
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΜΑΤΙΚΗΣ

# ΝΙΚΟΛΑΟΣ ΔΗΜΗΤΡΑΚΟΠΟΥΛΟΣ

2<sup>η</sup> Εργασία στο μάθημα **Λειτουργικά Συστήματα**

Ταύρος, 29 Ιανουαρίου 2020

## Περιεχόμενα

|   |    |
|---|----|
| Κώδικας εργασίας.....   | 3  |
| Server .....  | 3  |
| Server.h.....   | 3  |
| Server.c .....  | 3  |
| Tokenizer.h .....   | 18 |
| Tokenizer.c.....  | 19 |
| Client .....  | 23 |
| Client.h.....   | 23 |
| Client.c .....  | 24 |
| Ειδικές αναφορές σε ορισμένα σημεία.....  | 32 |
| Μέθοδος αποστολής / παραλαβής δεδομένων .....   | 32 |
| Τα Structs που μοντελοποιούν τις εντολές του χρήστη .....   | 32 |
| Ενδεικτικές Εκτελέσεις ( Screenshots ).....   | 34 |
| Server / Client επικοινωνία με διαγνωστικά μηνύματα και έλεγχος τερματισμού επικοινωνίας – PIN .....  | 34 |
| Παρατηρήσεις / Σχόλια .....   | 34 |
| Υποστήριξη πολλών Client.....   | 35 |
| Παρατηρήσεις / Σχόλια .....   | 35 |
| Ομαλή λειτουργία Server/Client (διαχείριση θυγατρικών διεργασιών - μη ύπαρξη Zombie - τερματισμός νέων διεργασιών στο κλείσιμο της επικοινωνίας με τον Client)..... | 36 |
| Παρατηρήσεις / Σχόλια .....   | 36 |
| Υποστήριξη απλών εντολών και αναφορά σφαλμάτων .....  | 37 |
| Παρατηρήσεις / Σχόλια .....   | 37 |
| Υποστήριξη εντολών με παραμέτρους και ορίσματα και αναφορά σφαλμάτων.....   | 38 |
| Παρατηρήσεις / Σχόλια .....   | 38 |
| Υποστήριξη πολλαπλών σωληνώσεων και αναφορά σφαλμάτων .....   | 39 |
| Παρατηρήσεις / Σχόλια .....   | 39 |
| Υποστήριξη απλών ανακατευθύνσεων και αναφορά σφαλμάτων.....   | 40 |
| Παρατηρήσεις / Σχόλια .....   | 40 |
| Υποστήριξη cd, history και exit.....  | 41 |
| Παρατηρήσεις / Σχόλια .....   | 41 |
| Γενικά Σχόλια / Παρατηρήσεις .....  | 42 |
| Με δυσκόλεψε / Δεν υλοποίησα .....  | 42 |
| Συνοπτικός Πίνακας .....  | 43 |

## Κώδικας εργασίας

---

### Server

#### Server.h

```
//  
// Created by delta on 3/1/20.  
//  
  
#include <stdio.h>  
#include "tokenizer.h"  
  
#ifndef SHELL_SERVER_H  
#define SHELL_SERVER_H  
  
//Basic Server functions  
int initiate_server(unsigned short);  
int connection_handler(int, int);  
  
//Execute commands  
int exec_pipes(int, cmdHolder*);  
  
//Custom implementations of recv and send  
ssize_t recv_all(int, char *, int);  
ssize_t send_all(int, char *, ssize_t);  
  
//Send fds to desired destinations  
ssize_t sendall_pipe(int, int);  
int sendall_pipe_file(int, int, char*);  
  
//Signal handlers  
void sigint_handler(int);  
void sigchld_handler(int);  
  
#endif //SHELL_SERVER_H
```

#### Server.c

```

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <signal.h>
#include <asm/ioctls.h>
#include <sys/ioctl.h>
#include <errno.h>
#include <fcntl.h>
#include "tokenizer.h"

#define BUFFER_SIZE 4096
#define MAX_CHUNK 4096
int server_sock;

int client_connections = 0;

#define PIN_SUCCESS 0
#define PIN_FAILURE 1

#define EMPTY_FD (-2)

void sigint_handler(int signum)
{
    close(server_sock);
    puts("\nServer main process terminated");
    exit(EXIT_SUCCESS);
}

void sigchld_handler(int signum)
{
    int status;
    //Harvest all zombies
    while(waitpid(-1, &status, WNOHANG) > 0);
}

ssize_t recv_all(int sockfd, char *buf, int flags)
{
    /*IN THIS VERSION OF recv_all WE EXPECT THE READ SIZE
    *NOT TO EXCEED THE BUFFER_SIZE CONSTANT. IF IT DOES,
    * SOMETHING WENT WRONG*/
    ssize_t received_bytes;

    //Read the total expected transmission size
    ssize_t toread = 0;
    received_bytes = read(sockfd, &toread, sizeof(toread));

```

```

    if (received_bytes <= 0)
        return received_bytes;

    //printf("Preparing to receive %ld bytes total\n", toread);

    ssize_t totalread = 0;

    //Continue reading from socket until there are bytes left to
read
    while (toread > 0)
    {
        //Read the chunk that the other end will attempt to send
        size_t chunk_size;
        received_bytes = read(sockfd, &chunk_size,
sizeof(chunk_size));

        if(received_bytes <= 0)
            return received_bytes;

        //If the expected chunk is bigger than the buffer size
then something went wrong
        if(chunk_size > BUFFER_SIZE)
        {
            fprintf(stderr, "Incompatible chunk size. Something
went wrong.");
            return -1;
        }

        bzero(buf, BUFFER_SIZE + 1);
        received_bytes = recv(sockfd, buf, chunk_size, flags);

        if (received_bytes <= 0)
            return received_bytes;

        totalread += received_bytes;
        toread -= received_bytes;
    }

    return totalread;
}

ssize_t send_all(int socket, char *buf, ssize_t size)
{
    ssize_t total_sent = 0; // how many bytes we've sent
    ssize_t sent_bytes;

    //Inform client for the total_sent size of the transmission
    ssize_t total_size = size;
    //printf("Notifying client that I want to send %ld bytes total
\n", size);
    sent_bytes = write(socket, &total_size, sizeof(total_size));

    if(sent_bytes <= 0)

```

```

        return sent_bytes;

    ssize_t transmission_size;
    //While all bytes have not been sent
    while(total_sent < size)
    {
        /*CHUNK SIZE
        * If the remaining data are bigger than our max chunk
size then send MAX_CHUNK
        * Else send the smaller remaining portion
        */
        if((size - total_sent) > MAX_CHUNK )
            transmission_size = MAX_CHUNK;
        else
            transmission_size = size - total_sent;

        //Send the decided chunk size to the client
        //printf("Notifying client that I want to send a %ld byte
chunk\n",transmission_size);
        sent_bytes = write(socket,&transmission_size,
sizeof(transmission_size));

        if(sent_bytes <= 0)
            return sent_bytes;

        //Send the chunk to the client
        sent_bytes = send(socket, buf + total_sent,
transmission_size, 0);
        //printf("Sent %ld bytes\n", sent_bytes);

        if (sent_bytes <= 0)
            return sent_bytes;

        total_sent += sent_bytes;
    }
    return total_sent;
}

ssize_t sendall_pipe(int socket, int fd )
{
    /*THE BASIC VERSION OF THE PIPE SENDING FUNCTION*/

    ssize_t sent_bytes;
    ssize_t total_sent = 0;
    int count = -1;
    char byte;
    char* buff;

    //While the file descriptor contains at least 1 byte
    while (read(fd, &byte, 1) == 1)
    {
        //Find the size of the file descriptor and store it to
count

```

```

        if (ioctl(fd, FIONREAD, &count) != -1)
        {
            //fprintf(stdout, "Preparing to send %d bytes\n",
count + 1);
            //Allocate the right-sized buffer
            buff = malloc(count + 1);
            //Store the first byte read
            buff[0] = byte;
            //Read all the remaining bytes
            if (read(fd, buff + 1, count) == count)
            {
                //Send all the buffer through sockets
                sent_bytes = send_all(socket, buff, count+1);
                if (sent_bytes <= 0)
                    return sent_bytes;

                total_sent += sent_bytes;
            }
            free(buff);
        }
    }
    if(count < 0)
        return EMPTY_FD;
    else
        return total_sent;
}

int sendall_pipe_file(int socket, int fd, char* filename)
{
    /*AN ALTERATION OF sendall_pipe. INSTEAD OF SENDING
    *THE FILE DESCRIPTOR THROUGH SOCKETS, IT IS STORED
    *IN A FILE.*/

    ssize_t sent_bytes;

    //Create the specified file
    int filefd = open(filename, O_WRONLY | O_CREAT, 0644);
    if(filefd == -1)
    {
        perror("open()");

        char *error_msg = strerror(errno);
        sent_bytes = send_all(socket, error_msg,
strlen(error_msg));

        if(sent_bytes <= 0)
            return -1;

        return EXIT_FAILURE;
    }

    int count;
    char byte;

```

```

char* buff;
while (read(fd, &byte, 1) == 1)
{
    if (ioctl(fd, FIONREAD, &count) != -1)
    {
        //fprintf(stdout, "Preparing to send %d bytes\n",
count + 1);
        buff = malloc(count + 1);
        buff[0] = byte;
        if (read(fd, buff + 1, count) == count)
        {
            //Write data to the file
            ssize_t written_bytes = write(filefd, buff, count
+ 1);

            //fprintf(stdout, "Wrote %zi bytes to
file\n", written_bytes);

            if(written_bytes < count)
            {
                perror("write()");

                char *error_msg = strerror(errno);
                sent_bytes = send_all(socket, error_msg,
strlen(error_msg));

                if(sent_bytes <= 0)
                    return -1;

                return EXIT_FAILURE;
            }

            sent_bytes = send_all(socket, "File made
successfully.", 23);
            if(sent_bytes <= 0)
                return -1;
        }
        free(buff);
    }
}
//Close the file
close(filefd);
return EXIT_SUCCESS;
}

/*Main idea taken from
https://stackoverflow.com/questions/17630247/coding-multiple-pipe-in-c
*But modified to match the command parser needs and some other
requirements*/
int exec_pipes(int fd, cmdHolder *commands)
{
    int  fds[2];
    pid_t pid;

```



```

int  fd_in = fd;

int current_cmd = 0;

char** args;
//While there are arguments left and are not redirection
commands
while ((args = get_arguments(commands, current_cmd) ) != NULL
&& get_commandType(commands,current_cmd) == PIPED )
{
    pipe(fds);
    int status;
    switch(pid = fork())
    {
        case -1: //fork() failed
            perror("fork()");
            exit(EXIT_FAILURE);
        case 0: //Child
            dup2(fd_in, 0); //Change the input according to
the old one

            char** next_args = get_arguments(commands,
current_cmd + 1);
            commandType next_type =
get_commandType(commands,current_cmd + 1);

            //If there is a next command and is PIPED
            if (next_args != NULL && next_type == PIPED)
                dup2(fds[1], 1);

            close(fds[0]);

            execvp(args[0], args);

            perror("Execvp()");
            exit(errno);

        default: //Parent
            waitpid(pid,&status,0);

            if (WIFEXITED(status))
            {
                int res = WEXITSTATUS(status);
                if(res != 0)
                {
                    /*IF A COMMAND DIDN'T EXECUTE SUCCESSFULLY
THEN
                    *INFORM THE CLIENT THAT SOMETHING WENT
WRONG

                    *ON THAT COMMAND.*/
                    char* error_msg = strerror(res);
                    printf("execvp() executing %s :
%s\n",args[0],error_msg);

```

```

        }
    }

    close(fds[1]);
    fd_in = fds[0]; //save the input for the next
command
    current_cmd++;
    break;
}
}
return EXIT_SUCCESS;
}

int connection_handler(int sockfd, int client_id)
{
    char client_message[BUFFER_SIZE];

    ssize_t sent_bytes;

    /*SIMPLE (AND COMPLETELY INSECURE) PASSWORD PROTOCOL
    * 1. SERVER INFORMS CLIENTS ABOUT THEIR UNIQUE CLIENT ID (SO
    THE USER CAN FIND THE CORRECT PASS FOR THEIR CLIENT INSTANCE)
    * 2. A RANDOM PIN IS GENERATED ON THE SERVER (0000-9999)
    * ONLY THE SERVER KNOWS THE PASSWORD
    * 3. CLIENT CAN MAKE ATTEMPTS UNTIL THEY ENTER THE CORRECT
    PASS (BRUTE-FORCE LOVERS SMILE HERE ONLY 10^4)
    * 4. SERVER CHECKS EVERY ATTEMPT AND RETURNS A FLAG OF
    SUCCESS OR FAILURE TO THE CLIENT
    * 5. WHEN THE CORRECT PIN IS RECEIVED BOTH SERVER AND CLIENT
    ARE READY TO DO THEIR JOBS
    */

    sent_bytes = write(sockfd, &client_id, sizeof(client_id));
    if (sent_bytes <= 0)
        return EXIT_FAILURE;

    srand((unsigned int) time(NULL));
    int PIN = rand() % 10000 ;

    int received_PIN;
    printf("Client #%d must use %04d as PIN to login\n",client_id,
PIN);
    do
    {
        ssize_t received_bytes = read(sockfd, &received_PIN,
sizeof(received_PIN));
        if (received_bytes <= 0)
            return EXIT_FAILURE;

        int answer;
        if(received_PIN != PIN)
        {

```

```

        printf("Client #%d entered wrong PIN
number\n",client_id);
        answer = PIN_FAILURE;
    }
    else
    {
        printf("Client #%d entered correct PIN
number\n",client_id);
        answer = PIN_SUCCESS;
    }
    sent_bytes = write(sockfd, &answer, sizeof(answer));

    if (sent_bytes <= 0)
        return EXIT_FAILURE;

}while(received_PIN != PIN);

while(true)
{
    //Receive the message from the client
    ssize_t read_size = recv_all(sockfd, client_message, 0);
    if(read_size <= 0)
        return EXIT_FAILURE;

    printf("Received from client #%d (%zd bytes): %s\n",
client_id,read_size,client_message);

    //Tokenize
    const char delimiters[] = {' ', '\n', '\t', '\0'};
    cmdHolder *cmds = tokenizer(client_message, delimiters);

    //If the struct doesn't contain any commands
    if(cmdHolder_isEmpty(cmds))
    {
        send_all(sockfd, "Could not parse any commands. Did
you type anything?", 53);
    }
    //If the client has pressed only exit
    else if ( strcmp(get_arguments(cmds, 0)[0], "exit") == 0 )
    {
        if(cmds->current_size == 1 &&
cmds->commands[0]->current_args_size == 1 )
        {
            printf("Client #%d requested exit. Process: %d\n",
client_id, getpid());

            int shut_res = close(sockfd);
            if (shut_res < 0)
            {
                perror("Socket shutdown failed");
                return EXIT_FAILURE;
            }
            return EXIT_SUCCESS;
        }
    }
}

```

```

    }
    else
        send_all(sockfd, "exit: Syntax error: no arguments
or other commands must be provided", 68);
    }
    //If the client has typed cd with 1 argument
    else if ( strcmp(get_arguments(cmds, 0)[0], "cd") == 0 )
    {
        if(cmds->current_size == 1 &&
            cmds->commands[0]->current_args_size == 2 )
        {
            //Change the directory
            int exit_code = chdir(get_arguments(cmds, 0)[1]);
            if (exit_code < 0)
            {
                //If error occurs send the error message
                through sockets
                char *error_msg = strerror(errno);
                send_all(sockfd, error_msg, strlen(error_msg)
+ 1);
            } else
                send_all(sockfd, "Successfully changed
directory", 31);
        }
        else
            send_all(sockfd, "cd: Syntax error: 1 argument and
no other commands must be provided", 68);
    }

    //Execute the supplied command(s) as exec-supported
    else
    {
        //Storing the result of exec_pipes through piping

        int fds[2];
        pipe(fds);

        int pid;
        switch (pid = fork())
        {
            case -1: //Fork error
            {
                perror("fork()");
                return EXIT_FAILURE;
            }
            case 0: //Child
            {
                dup2(fds[1], 1);
                close(fds[0]);

                int exit_code = exec_pipes(fds[1], cmds);

                /*When all commands are successfully executed,

```

```

a 0 is returned from exec_pipes.
    * If something else has been returned then
something went wrong.
    * So we shall give the child the proper exit
code depending on the exec_pipes
    * return code so as to notify the parent. */

close(fds[1]);

if (exit_code != 0)
    exit(EXIT_FAILURE);
else
    exit(EXIT_SUCCESS);

}
default: //Parent
{
    dup2(fds[0], 0);
    close(fds[1]);

    /*Waiting for the child to finish executing
(taking care of zombies)
    * and simultaneously getting the exit_code of
the child*/

    int child_status;
    int res = waitpid(pid, &child_status, 0);

    if (res < 0)
        perror("waitpid()");

    //If the child has terminated in an
anticipated fashion
    if (WIFEXITED(child_status))
    {
        //If the status returned is 0
        if (WEXITSTATUS(child_status) == 0)
            printf("Command executed successfully.
Ready for transmission\n");
        else
            fprintf(stderr, "Command executed with
errors. Ready to transmit error message\n");
    }
    else
    {
        fprintf(stderr, "Fatal error occured.
Exiting abnormally...\n");
        return EXIT_FAILURE;
    }

    /*Checking if the command has redirection.
    * As a reminder, redirection is allowed only
at the end.

```

```

        * If given anywhere else but in the end, the
result of the command
        * before that redirection will be sent to the
client */

        if (cmdHolder_hasRedirection(cmds))
        {
            int redirection_result =
sendall_pipe_file(sockfd, fds[0], get_arguments(cmds, cmds-
>current_size - 1)[0]);

            switch(redirection_result)
            {
                case EXIT_SUCCESS:
                    fprintf(stdout, "Client #d:
Redirection completed successfully.\n", client_id);
                    break;
                case EXIT_FAILURE:
                    fprintf(stderr, "Client #d:
Redirection completed with errors.\n", client_id);
                    break;
                case -1:
                    fprintf(stderr, "Client #d: Error
in transmission.\n", client_id);
                    return EXIT_FAILURE;
                default:
                    fprintf(stderr, "Implementation
error: Undefined scenario.\n");
            }
        }
        else
        {

            /*TRANSMIT THE COMMAND RESULT TO THE
CLIENT
            *IF THE COMMAND HAD OUTPUT THEN SEND IT
THROUGH SOCKETS
            *BUT SOME COMMANDS (e.g rm) DO NOT
PRODUCE OUTPUT EVEN IN
            *SUCCESSFUL EXECUTION. IN THOSE CASES A
\0 BECAUSE THE CLIENT
            *MUST RECEIVE SOMETHING*/

            sent_bytes = sendall_pipe(sockfd, fds[0]);
            switch(sent_bytes)
            {
                case EMPTY_FD:
                {
                    sent_bytes = send_all(sockfd, "",
1);

                    if (sent_bytes <= 0)
                    {

```

```

        fprintf(stderr, "Client #d:
Error in transmission.\n", client_id);
        return EXIT_FAILURE;
    }
    break;
}
default:
{
    if (sent_bytes <= 0)
    {
        fprintf(stderr, "Client #d:
Error in transmission.\n", client_id);
        return EXIT_FAILURE;
    }
}
}
}
}
close(fds[0]);
break;
}
}
}
//Free the memory allocated for the structures
free_cmdHolder(cmds);
}
}

int initiate_server(unsigned short port)
{
    struct sockaddr_in server;
    struct sockaddr_in client;
    int socklength = sizeof(struct sockaddr_in);

    //Create socket
    server_sock = socket(AF_INET , SOCK_STREAM , 0);
    if (server_sock < 0)
    {
        perror("Socket");
        exit(EXIT_FAILURE);
    }

    //Prepare the sockaddr_in structure
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons( port );

    //Bind
    if(bind(server_sock, (struct sockaddr *)&server ,
sizeof(server)) < 0)
    {
        perror("Bind");
        exit(EXIT_FAILURE);
    }
}

```

```

else
    puts("Bind done");

//Listen
if(listen(server_sock , 5) < 0 )
{
    perror("listen()");
    exit(EXIT_FAILURE);
}

else
    puts("Waiting for incoming connections...");

//Accept new connections
int new_socket;
while((new_socket = accept(server_sock, (struct sockaddr
*)&client, (socklen_t*)&socklength) ) > 0 )
{
    //If something went wrong in the accept process
    if (new_socket < 0)
    {
        perror("Connection accept failed");
        exit(EXIT_FAILURE);
    }

    //Increment total client connections since the start of
server's execution
    client_connections++;

    //Get info about the client
    char *client_ip = inet_ntoa(client.sin_addr);
    int client_port = ntohs(client.sin_port);

    printf("Connection accepted from IP: %s port: %d Client
%d\n", client_ip, client_port, client_connections);

    //Multiple clients support
    int pid;
    int client_id;
    switch(pid = fork())
    {
        case -1: //Fork error
        {
            perror("fork()");
            exit(EXIT_FAILURE);
        }
        case 0: //Child
        {
            //Give the client a unique ID
            client_id = client_connections;

            //Execute the handler for that client
            int exit_status = connection_handler(new_socket,

```



```

client_id);

        printf("Client #%d disconnected : Address: %s
port: %d\n", client_id, client_ip, client_port);

        switch(exit_status)
        {
            case EXIT_SUCCESS:
                printf("Client #%d exited properly
(exit).\n",client_id);
                break;
            case EXIT_FAILURE:
                fprintf(stderr,"Client #%d exited
abnormally.\n",client_id);
        }
        exit(EXIT_SUCCESS);
    }
    default: //Parent
    {
        /*We cannot wait here for the child to finish
because that would hang the accept of new clients.
* Instead, to prevent zombies, the program relies
on the sigchld_handler when SIGCHLD is triggered
* to do the job. */
        printf("Handler assigned by creating process
%d\n", pid);
        break;
    }
}
}
exit(EXIT_SUCCESS);
}

int main(int argc , char *argv[])
{
    if(argc != 2)
    {
        fprintf(stderr,"Usage: ./server [port]\n");
        exit(EXIT_FAILURE);
    }

    //Convert port from string to integer
    errno = 0;
    char* strtol_check = NULL;
    long int port = strtol(argv[1],&strtol_check,10);
    //Check for validity of the port
    if((port == 0 && errno != 0 ) ||
        argv[1] == strtol_check || //From documentation
        port <= 0 || port > 65535)
    {
        fprintf(stderr,"Invalid port argument (1-65535 only)\n");
        exit(EXIT_FAILURE);
    }
}

```

```

    //Replacing the default handlers
    signal(SIGCHLD,sigchld_handler);
    signal(SIGINT,sigint_handler);

    //Start the server
    initiate_server((unsigned short)port);
}

```

Tokenizer.h

```

//
// Created by delta on 20/1/20.
//

#ifndef TOKENIZER_TOKENIZER_H
#define TOKENIZER_TOKENIZER_H

#include <stdbool.h>

//Command type
typedef enum
{
    PIPED = 0,
    OUT_REDIRECTION = 1
}commandType;

//Struct that holds the arguments of a command
struct command
{
    commandType type ;
    char** arguments;
    int current_args_size;
    int args_capacity;
};
typedef struct command cmd;

//Command struct functions
cmd* cmd_initialize(commandType);
void cmd_realloc(cmd*);
bool cmd_isAlmostFull(cmd*);
void cmd_addArgument(char *token, cmd *n);
void cmd_nullify(cmd*);
void free_cmd(cmd*);

//Struct that holds commands (command structs)
struct commands_holder
{
    cmd** commands;
    int current_size;
}

```

```

    int capacity;
};
typedef struct commands_holder cmdHolder;

//commands_holder struct functions

cmdHolder* cmdHolder_initialize(void);
void cmdHolder_realloc(cmdHolder *);
bool cmdHolder_isAlmostFull(cmdHolder *);
void add_cmd(cmdHolder*, cmd*);
void cmdHolder_nullify(cmdHolder *);
char** get_arguments(cmdHolder*, int);
commandType get_commandType(cmdHolder*, int);
void free_cmdHolder(cmdHolder *);
bool cmdHolder_isEmpty(cmdHolder *);
bool cmdHolder_hasRedirection(cmdHolder *);
cmdHolder * tokenizer(char*, const char*);

#endif //TOKENIZER_TOKENIZER_H

```

Tokenizer.c

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include "tokenizer.h"

cmd* cmd_initialize(commandType type)
{
    cmd* n = malloc(sizeof(cmd));

    n->type = type;
    n->current_args_size = 0;
    n->args_capacity = 5;
    n->arguments = malloc(n->args_capacity * sizeof(char*));
    if(n->arguments == NULL)
    {
        fprintf(stderr, "Could not allocate memory, exiting.\n");
        exit(EXIT_FAILURE);
    }
    cmd_nullify(n);
    return n;
}

void cmd_realloc(cmd* n)
{
    n->args_capacity *= 2;
    n->arguments = realloc(n->arguments, n->args_capacity *
sizeof(char*));
    if(n->arguments == NULL)

```

```

    {
        fprintf(stderr, "Could not allocate memory, exiting.\n");
        exit(EXIT_FAILURE);
    }
    cmd_nullify(n);
}
bool cmd_isAlmostFull(cmd* n)
{
    if(n->current_args_size + 1 >= n->args_capacity)
        return true;
    else
        return false;
}
void cmd_addArgument(char* token, cmd* n)
{
    if(cmd_isAlmostFull(n))
        cmd_realloc(n);

    n->arguments[n->current_args_size++] = token;
}
void cmd_nullify(cmd* n)
{
    for(int i = n->current_args_size ; i < n->args_capacity ; i++)
        n->arguments[i] = NULL;
}
void free_cmd(cmd* n)
{
    free(n->arguments);
    free(n);
}

cmdHolder* cmdHolder_initialize()
{
    cmdHolder* n = malloc(sizeof(cmdHolder));

    n->current_size = 0;
    n->capacity = 5;
    n->commands = malloc(n->capacity * sizeof(cmd*));
    if(n->commands == NULL)
    {
        fprintf(stderr, "Could not allocate memory, exiting.\n");
        exit(EXIT_FAILURE);
    }
    cmdHolder_nullify(n);
    return n;
}
void cmdHolder_realloc(cmdHolder * n)
{
    n->capacity *= 2;
    n->commands = realloc(n->commands, n->capacity *
sizeof(cmd*));
    if(n->commands == NULL)

```

```

{
    fprintf(stderr, "Could not allocate memory, exiting.\n");
    exit(EXIT_FAILURE);
}
cmdHolder_nullify(n);
}
bool cmdHolder_isAlmostFull(cmdHolder * n)
{
    if(n->current_size + 1 >= n->capacity)
        return true ;
    else
        return false;
}
void add_cmd(cmdHolder* n , cmd* command)
{
    if(cmdHolder_isAlmostFull(n))
        cmdHolder_realloc(n);

    n->commands[n->current_size++] = command;
}
void cmdHolder_nullify(cmdHolder* n)
{
    for(int i = n->current_size ; i < n->capacity ; i++)
        n->commands[i] = NULL;
}
char** get_arguments(cmdHolder* n, int index)
{
    if(n->commands[index] != NULL)
        return n->commands[index]->arguments;
    else
        return NULL;
}
commandType get_commandType(cmdHolder* n, int index)
{
    if(n->commands[index] != NULL)
        return n->commands[index]->type;
    else
        return -1;
}
void free_cmdHolder(cmdHolder* n)
{
    for(int i = 0 ; i < n->current_size ; i++)
        free_cmd(n->commands[i]);
    free(n->commands);

    free(n);
}
bool cmdHolder_isEmpty(cmdHolder* n)
{
    if(n->current_size == 0)
        return true;
    else
        return false;
}

```

```

}
bool cmdHolder_hasRedirection(cmdHolder* n)
{
    int index = 0;
    while(n->commands[index] != NULL)
    {
        if (get_commandType(n,index) == OUT_REDIRECTION)
        {
            if (index == n->current_size - 1)
                return true;
            else
            {
                fprintf(stderr, "Implementation error: output
redirection allowed only at the last command\n");
                return false;
            }
        }
        index++;
    }
    return false;
}

cmdHolder * tokenizer(char* string, const char* delimiters)
{
    //The first command is considered PIPED by default
    cmdHolder* command_holder = cmdHolder_initialize();
    cmd* command = cmd_initialize(PIPED);

    char *token;

    token = strtok(string, delimiters);

    //If no tokens are found, then return an empty cmdHolder
    if(token == NULL)
        return command_holder;

    /* walk through other tokens */
    while( token != NULL )
    {
        /*If a pipe or redirection is found
        *then store the command in the command_holder
        *and initialize a new command*/
        if(strcmp(token,"|") == 0)
        {
            add_cmd(command_holder, command);
            command = cmd_initialize(PIPED);
            token = strtok(NULL, delimiters);
            continue;
        }
        else if(strcmp(token,">") == 0)

```

```

    {
        add_cmd(command_holder, command);
        command = cmd_initialize(OUT_REDIRECTION);
        token = strtok(NULL, delimiters);
        continue;
    }
    //Add arguments
    cmd_addArgument(token, command);

    token = strtok(NULL, delimiters);
}

/*When strtok doesn't have any more tokens
 *then add the last command to the command_holder*/

add_cmd(command_holder, command);

return command_holder;
}

```

## Client

### Client.h

```

//
// Created by delta on 3/1/20.
//

#include <stdio.h>

#ifndef SHELL_CLIENT_H
#define SHELL_CLIENT_H

//Start the client
void initiate_client(char*, unsigned short);

//Signal handlers
void sigpipe_handler(int);
void sigint_handler(int);

//Custom implementations of recv and send
ssize_t recv_all(int, char**, int);
ssize_t send_all(int, char*, ssize_t);

//Handling user input
void input_handler(char*, int, char*);

//Print the history file
void create_history();

```

```
void print_history();
void delete_history();

#endif //SHELL_CLIENT_H
```

Client.c

```
#include "Client.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <signal.h>
#include <stdbool.h>
#include <fcntl.h>
#include <errno.h>

#define BUFFER_SIZE 4096

int clientSocket;
FILE* file;

#define PIN_SUCCESS 0
#define PIN_FAILURE 1

void sigpipe_handler(int signum)
{
    delete_history();
    close(clientSocket);
    puts("SIGPIPE triggered");
    exit(EXIT_FAILURE);
}

void sigint_handler(int signum)
{
    delete_history();
    close(clientSocket);
    printf("\nClient terminated\n");
    exit(EXIT_SUCCESS);
}

ssize_t recv_all(int socket, char **server_response, int flags)
{
    ssize_t received_bytes;

    //Get the total size of the transmission
    ssize_t size = 0;
```



```

received_bytes = read(socket, &size, sizeof(size));
//printf("Preparing to receive %ld bytes total\n",size);
if (received_bytes <= 0)
    return received_bytes;

/* THIS IMPLEMENTATION COULD ALSO WORK WITH A STATICALLY
 * ALLOCATED BUFFER THAT HAS SIZE EQUAL TO THE CHUNK DATA + 1.
 * THE DIFFERENCE IS THAT THE BUFFER MUST BE PRINTED AFTER
 * EACH CHUNK IS READ SUCCESSFULLY*/

//Allocate a buffer big enough for the data to fit
*server_response = malloc((size+1) * sizeof(char));
bzero(*server_response, size + 1);

ssize_t toread = size;
ssize_t total_read = 0;

//While there are data left to be transmitted
while (toread > 0)
{
    //Get chunk_size from server
    ssize_t chunk_size;
    received_bytes = read(socket, &chunk_size,
sizeof(chunk_size));
    //printf("chunk_size = %ld\n",chunk_size);
    if (received_bytes <= 0)
        return received_bytes;

    //Read the whole chunk
    ssize_t chunk_read = 0;
    while(chunk_read < chunk_size)
    {
        //Receive the chunk data
        received_bytes = recv(socket, *server_response +
total_read, chunk_size - chunk_read, flags);
        //printf("Read %ld bytes\n",received_bytes);
        if (received_bytes <= 0)
            return received_bytes;

        chunk_read += received_bytes;
        total_read += received_bytes;
        //printf("Total chunk read = %ld\n",chunk_read);
    }
    toread -= chunk_read;
}
return total_read;
}

ssize_t send_all(int socket, char *buf, ssize_t size)
{
    /* LIKE THE SERVER VERSION */

    ssize_t total_bytes = 0;           // how many bytes we've sent

```

```

    ssize_t bytes_sent;

    //printf("Notifying server that I want to send %ld bytes total\n",size);
    send(socket,&size, sizeof(size),0); //FOR CLIENT

    ssize_t chunk_size;
    while(total_bytes < size)
    {
        chunk_size = size - total_bytes;

        //printf("Notifying server that I want to send a %ld byte chunk\n",chunk_size);
        bytes_sent = send(socket, &chunk_size, sizeof(chunk_size),0);

        if(bytes_sent <= 0)
            return bytes_sent;

        bytes_sent = send(socket, buf + total_bytes, chunk_size,0);

        //printf("Sent %ld bytes\n",n);
        if (bytes_sent <= 0)
            return bytes_sent;

        total_bytes += bytes_sent;
    }
    return total_bytes;
}

void input_handler(char* buffer, int buf_size, char* ip_addr)
{
    //Gets user input
    bzero(buffer, sizeof(buf_size));
    printf("\n%s_>: ",ip_addr);
    int n = 0;
    while ((buffer[n++] = getchar()) != '\n');
    buffer[n - 1] = '\0';
}

void create_history()
{
    /*Filename is unique for each client instance on
    *the same machine thanks to the PID*/

    char filename[30];
    sprintf(filename,"his%d",getpid());

    //Create history file
    file = fopen(filename,"w+");
    if(file == NULL)
    {
        perror("fopen()");
    }
}

```

```

        exit(EXIT_FAILURE);
    }
}

void print_history()
{
    fseek(file,0,SEEK_SET);
    char c = fgetc(file);
    while (c != EOF)
    {
        printf ("%c", c);
        c = fgetc(file);
    }
}

void delete_history()
{
    int exit_code ;
    //Close the file if it has been created
    if(file != NULL)
    {
        exit_code = fclose(file);
        if(exit_code < 0)
            return;
    }

    char filename[30];
    sprintf(filename,"his%d",getpid());
    remove(filename);
}

void initiate_client(char* ip_address , unsigned short port)
{
    struct sockaddr_in serverAddr;
    socklen_t addr_size;

    clientSocket = socket(PF_INET, SOCK_STREAM, 0);
    if(clientSocket < 0)
    {
        perror("Socket");
        exit(EXIT_FAILURE);
    }

    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(port);
    serverAddr.sin_addr.s_addr = inet_addr(ip_address);
    memset(serverAddr.sin_zero, '\0', sizeof serverAddr.sin_zero);

    //Connect the socket to the server using the address struct
    addr_size = sizeof serverAddr;
    int status = connect(clientSocket, (struct sockaddr *)
&serverAddr, addr_size);
}

```

```

    if(status < 0)
    {
        close(clientSocket);
        perror("Connection failed");
        exit(EXIT_FAILURE);
    }
}

void authenticate_connection(char* ip_address, char* buffer)
{
    ssize_t received_bytes;

    //Get the client's unique identifier
    int client_id;
    received_bytes = read(clientSocket, &client_id,
sizeof(client_id));
    if(received_bytes <= 0)
    {
        printf("Connection terminated\n");
        exit(EXIT_SUCCESS);
    }

    printf("Client ID: %d\n", client_id);
    printf("Enter PIN code provided by the server\n");
    while(true)
    {
        input_handler(buffer, sizeof(buffer), ip_address);

        unsigned long input_length = strlen(buffer);
        errno = 0;
        char* strtol_check = NULL;

        //Convert string to int
        long int typed_value = strtol(buffer, &strtol_check, 10);

        //Check for validity of input
        if((typed_value == 0 && errno != 0 ) ||
            buffer == strtol_check || //From documentation
            typed_value < 0 || typed_value > 9999 ||
            input_length != 4)
        {
            printf("Invalid password format. Please enter a 4-
digit PIN code.\n");
            continue;
        }

        int PIN = (int)typed_value;
        //Send PIN to server
        ssize_t sent_bytes = write(clientSocket, &PIN,
sizeof(PIN));

        if(sent_bytes <= 0)
        {

```

```

        printf("Connection terminated\n");
        exit(EXIT_SUCCESS);
    }

    //Receive authentication answer from server
    int authentication_byte;
    received_bytes = read(clientSocket, &authentication_byte,
sizeof(authentication_byte));
    if(received_bytes <= 0)
    {
        printf("Connection terminated\n");
        exit(EXIT_SUCCESS);
    }

    if(authentication_byte == PIN_SUCCESS)
    {
        printf("Correct PIN code. You can now execute
commands.");
        break;
    }
    else if(authentication_byte == PIN_FAILURE)
        printf("Wrong PIN entered, please type again.");
    }
}

int connection_handler(char* ip_address , char *buffer)
{
    char *server_response = NULL;

    while(true)
    {
        ssize_t sent_bytes;

        //Get user's command
        input_handler(buffer, sizeof(buffer),ip_address);

        //If the command is history, print history and proceed to
next input
        if(strcmp(buffer,"history") == 0)
        {
            print_history();
            fprintf(file,"%s\n", buffer);
            continue;
        }

        //Send the command to the server
        sent_bytes = send_all(clientSocket, buffer, strlen(buffer)
+ 1);
        if(sent_bytes < 0)
        {
            puts("Connection has been terminated.");
            return EXIT_FAILURE;
        }
    }
}

```

```

        //If user typed exit then exit the handler
        if(strcmp(buffer,"exit") == 0)
            return EXIT_SUCCESS;

        //Store command to history file
        fprintf(file,"%s\n", buffer);

        //Receive the command result from server
        ssize_t received_bytes
=   recv_all(clientSocket,&server_response ,0);
        if(received_bytes <= 0)
        {
            puts("Connection has been terminated.");
            return EXIT_SUCCESS;
        }
        printf("%s",server_response);
        free(server_response);
    }
}

int main(int argc , char* argv[])
{
    if(argc != 3)
    {
        fprintf(stderr,"Usage: ./client [ip address] [port]\n");
        exit(EXIT_FAILURE);
    }

    char* ip_address = argv[1];
    long int port = strtol(argv[2],NULL,10);

    //Quick and dirty check
    if(port <= 0 || port > 65535 )
    {
        fprintf(stderr,"Invalid port argument (1-65535 only)\n");
        exit(EXIT_FAILURE);
    }

    signal(SIGPIPE, sigpipe_handler);
    signal(SIGINT, sigint_handler);

    char buffer[BUFFER_SIZE + 1];

    //Initiate connection
    initiate_client(ip_address,(unsigned short)port);
    //Authenticate Connection
    authenticate_connection(ip_address,buffer);

    //Create history file
    create_history();

```

```
//Handle the rest of the connection
connection_handler(ip_address,buffer);

//Delete history file
delete_history();

return EXIT_SUCCESS;
}
```

## Ειδικές αναφορές σε ορισμένα σημεία

### Μέθοδος αποστολής / παραλαβής δεδομένων

Οι `recv_all()` και `send_all()` σχεδιάστηκαν έτσι ώστε να μπορούν να στέλνουν δεδομένα οποιουδήποτε μεγέθους. Αυτό έγινε χρησιμοποιώντας την λογική των `chunks`, δηλαδή σπάζοντας ένα μεγάλο μήνυμα σε πολλά μικρότερα κομμάτια. Σχεδιάσα ένα απλό Application Layer πρωτόκολλο το οποίο έχει την εξής λογική:

1. Ο αποστολέας ενημερώνει τον παραλήπτη για το συνολικό μέγεθος της αποστολής που πρόκειται να κάνει
2. Έπειτα ο αποστολέας αρχίζει να σπάει τα δεδομένα σε `chunks`: αν το υπολειπόμενο μέγεθος είναι μεγαλύτερο από μέγιστο μέγεθος ενός `chunk`, τότε ο αποστολέας ενημερώνει τον παραλήπτη ότι πρόκειται να στείλει `chunk` ίσο με το `MAX_CHUNK`, αλλιώς τον ενημερώνει για το μέγεθος της μικρότερης ή ίσης ποσότητας.
3. Ο αποστολέας μετά μεταδίδει μέσω δικτύου το `chunk` αυτό και ο παραλήπτης λαμβάνει όσα πακέτα έχουν σχέση με αυτό (σε πραγματικές συνθήκες δικτύου **δεν** είναι εγγυημένο ότι όλο το `chunk` θα σταλθεί σε ένα πακέτο κι ας μην ξεπερνάνε τα δεδομένα το `MTU`).
4. Ο αποστολέας καταλαβαίνει ότι τελείωσε η παραλαβή των δεδομένων όταν το μέγεθος όλων των `chunks` που έχουν σταλθεί έχει γίνει ίσο με το συνολικό μέγεθος της αποστολής.

Διάφορες δοκιμές έχουν γίνει με πιο ακραίο παράδειγμα την αποστολή του περιεχομένου ενός αρχείου 50KB μέσω της εντολής `cat` μεταξύ 2 διαφορετικών υπολογιστών του τοπικού μου δικτύου. Δεν γνωρίζω αν ο αλγόριθμος αυτός θα παραμείνει σταθερός σε πιο ασταθείς περιπτώσεις δικτύων.

### Τα Structs που μοντελοποιούν τις εντολές του χρήστη

Για να γίνει δυνατή η πολλαπλή σωλήνωση και ανακατεύθυνση αλλά και για να διευκολυνθεί η διαχείριση της εισόδου, ακολουθήθηκε μια αφαιρετική προσέγγιση για να είναι δυνατός ο εύκολος διαχωρισμός της συντακτικής λογικής της εντολής του χρήστη.

Η απλή σωλήνωση όπως και η πολλαπλή, απαιτεί σαφή διαχωρισμό μεταξύ των εντολών που συμμετέχουν στη κάθε σωλήνωση. Οπότε είναι λογικό κανείς να σκεφτεί, χρειάζεται μία δομή που να κρατά όλα τα ορίσματα μιας εντολής. Έτσι λοιπόν δημιουργήθηκε το `struct command`, το οποίο έχει ως κύριο πεδίο ένα `char**` το οποίο θα δείχνει σε έναν `malloced` 2D πίνακα που θα κρατάει όλα τα `strings` που επιστρέφει η `strtok`. Επειδή όμως δεν μπορούμε να είμαστε σίγουροι για το πλήθος των `arguments` που μπορεί να έχει μια εντολή, πρέπει να μεριμνήσουμε ώστε να κάνουμε την διαδικασία του `parsing` όσο πιο δυναμική γίνεται. Με τη χρήση της `realloc()` λοιπόν μπορούμε δυναμικά να μεγαλώνουμε το μέγιστο πλήθος των `arguments` αν αυτό είναι απαραίτητο. Μια λεπτομέρεια σημαντική είναι ότι μετά την τελευταία παράμετρο του χρήστη, πρέπει υποχρεωτικά να υπάρχει τουλάχιστον μια τιμή που να έχει τη τιμή `NULL` ώστε η `execvp` να καταλαβαίνει που τελείωσαν τα ορίσματα.

Τα άλλα πεδία του `struct command` είναι βοηθητικά αλλά απολύτως χρήσιμα για την αποδοτική λειτουργία της δομής όπως θα παρατηρήσετε και στον κώδικα της εργασίας.

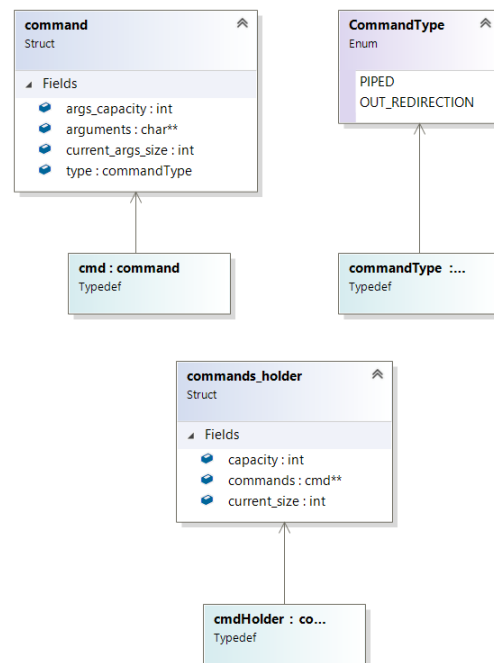


```
char *ls[] = {"ls", NULL};
char *grep[] = {"grep", "file", NULL};
char *wc[] = {"wc", NULL};
char **cmd[] = {ls, grep, wc, NULL};
```

**Εικόνα 0-1:** Η μορφή στην οποία θα έπρεπε να αποθηκεύονταν οι εντολές, αν δεν υπήρχε κάποια ειδική δομή.

Το επόμενο βήμα είναι να δημιουργηθεί μία δομή που θα κρατάει όλα τα struct command και θα επιτρέπει εύκολη προσπέλαση σε αυτά. Η λογική και οι συναρτήσεις που αλληλοεπιδρούν με αυτό το struct είναι εξαιρετικά παρόμοιες με το προηγούμενο οπότε δεν θα αναλυθούν περαιτέρω.

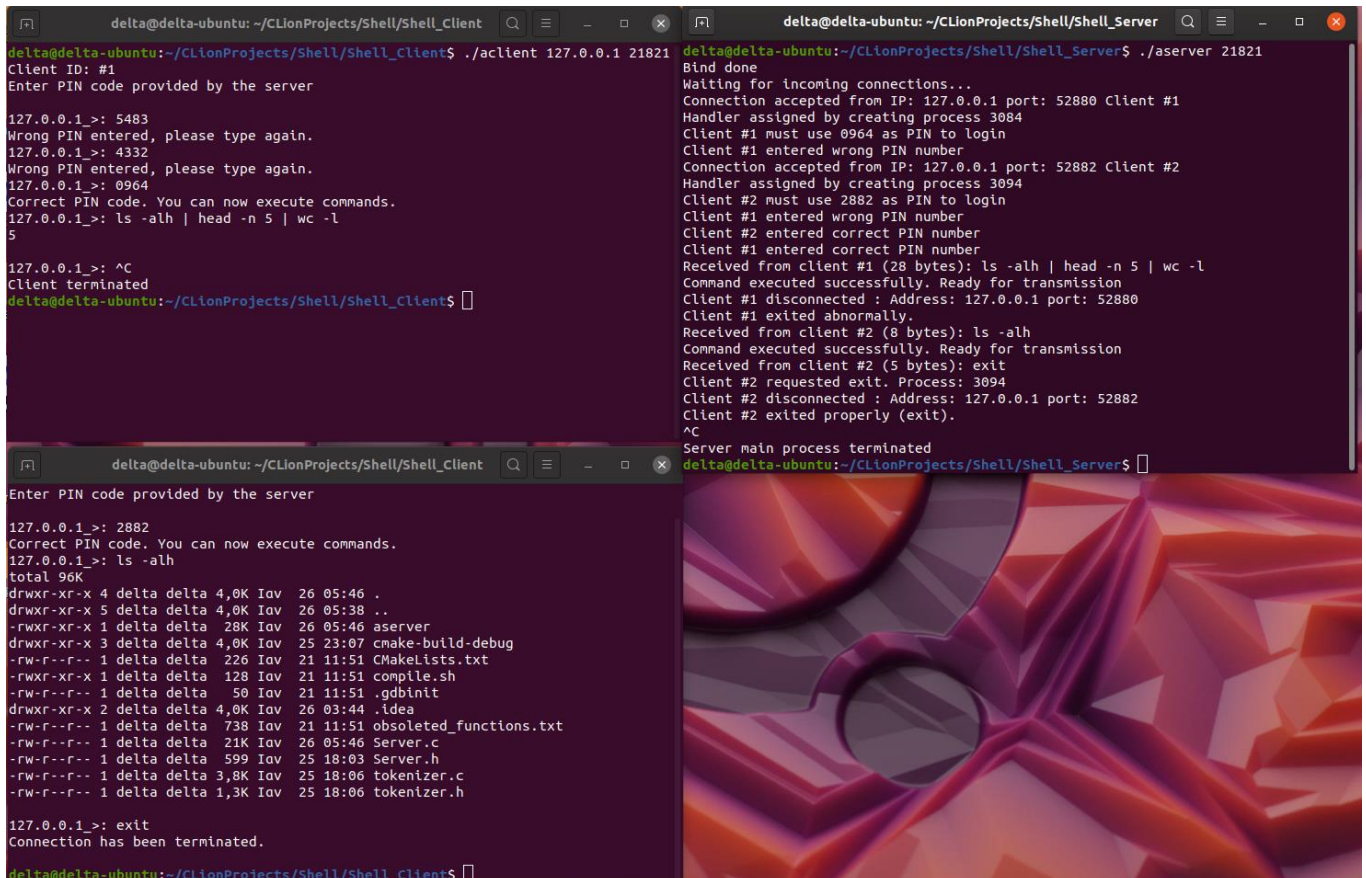
Αυτά τα 2 structs δένονται μεταξύ τους με την συνάρτηση tokenizer που παίρνει την είσοδο του χρήστη και την μετατρέπει σε ένα commands\_ holder struct που με τη σειρά του περιέχει command structs.



**Εικόνα 0-2:** Class diagram του Visual Studio που δείχνει τη δομή των 2 structs καθώς και το Enum CommandType, του οποίου η χρησιμότητα φαίνεται στην εφαρμογή των πολλαπλών σωληνώσεων και ανακατευθύνσεων.

## Ενδεικτικές Εκτελέσεις ( Screenshots )

Server / Client επικοινωνία με διαγνωστικά μηνύματα και έλεγχος τερματισμού επικοινωνίας – PIN



```
delta@delta-ubuntu: ~/CLionProjects/Shell/Shell_Client
delta@delta-ubuntu:~/CLionProjects/Shell/Shell_Client$ ./aclient 127.0.0.1 21821
Client ID: #1
Enter PIN code provided by the server

127.0.0.1_>: 5483
Wrong PIN entered, please type again.
127.0.0.1_>: 4332
Wrong PIN entered, please type again.
127.0.0.1_>: 0964
Correct PIN code. You can now execute commands.
127.0.0.1_>: ls -alh | head -n 5 | wc -l
5
127.0.0.1_>: ^C
Client terminated
delta@delta-ubuntu:~/CLionProjects/Shell/Shell_Client$

delta@delta-ubuntu:~/CLionProjects/Shell/Shell_Server
delta@delta-ubuntu:~/CLionProjects/Shell/Shell_Server$ ./aserver 21821
Bind done
Waiting for incoming connections...
Connection accepted from IP: 127.0.0.1 port: 52880 Client #1
Handler assigned by creating process 3084
Client #1 must use 0964 as PIN to login
Client #1 entered wrong PIN number
Connection accepted from IP: 127.0.0.1 port: 52882 Client #2
Handler assigned by creating process 3094
Client #2 must use 2882 as PIN to login
Client #1 entered wrong PIN number
Client #2 entered correct PIN number
Client #1 entered correct PIN number
Received from client #1 (28 bytes): ls -alh | head -n 5 | wc -l
Command executed successfully. Ready for transmission
Client #1 disconnected : Address: 127.0.0.1 port: 52880
Client #1 exited abnormally.
Received from client #2 (8 bytes): ls -alh
Command executed successfully. Ready for transmission
Received from client #2 (5 bytes): exit
Client #2 requested exit. Process: 3094
Client #2 disconnected : Address: 127.0.0.1 port: 52882
Client #2 exited properly (exit).
^C
Server main process terminated
delta@delta-ubuntu:~/CLionProjects/Shell/Shell_Server$

delta@delta-ubuntu:~/CLionProjects/Shell/Shell_Client
Enter PIN code provided by the server

127.0.0.1_>: 2882
Correct PIN code. You can now execute commands.
127.0.0.1_>: ls -alh
total 96K
drwxr-xr-x 4 delta delta 4,0K Iov 26 05:46 .
drwxr-xr-x 5 delta delta 4,0K Iov 26 05:38 ..
-rwxr-xr-x 1 delta delta 28K Iov 26 05:46 aserver
drwxr-xr-x 3 delta delta 4,0K Iov 25 23:07 cmake-build-debug
-rw-r--r-- 1 delta delta 226 Iov 21 11:51 CMakeLists.txt
-rwxr-xr-x 1 delta delta 128 Iov 21 11:51 compile.sh
-rw-r--r-- 1 delta delta 50 Iov 21 11:51 .gdbinit
drwxr-xr-x 2 delta delta 4,0K Iov 26 03:44 .idea
-rw-r--r-- 1 delta delta 738 Iov 21 11:51 obsoleted_functions.txt
-rw-r--r-- 1 delta delta 21K Iov 26 05:46 Server.c
-rw-r--r-- 1 delta delta 599 Iov 25 18:03 Server.h
-rw-r--r-- 1 delta delta 3,8K Iov 25 18:06 tokenizer.c
-rw-r--r-- 1 delta delta 1,3K Iov 25 18:06 tokenizer.h

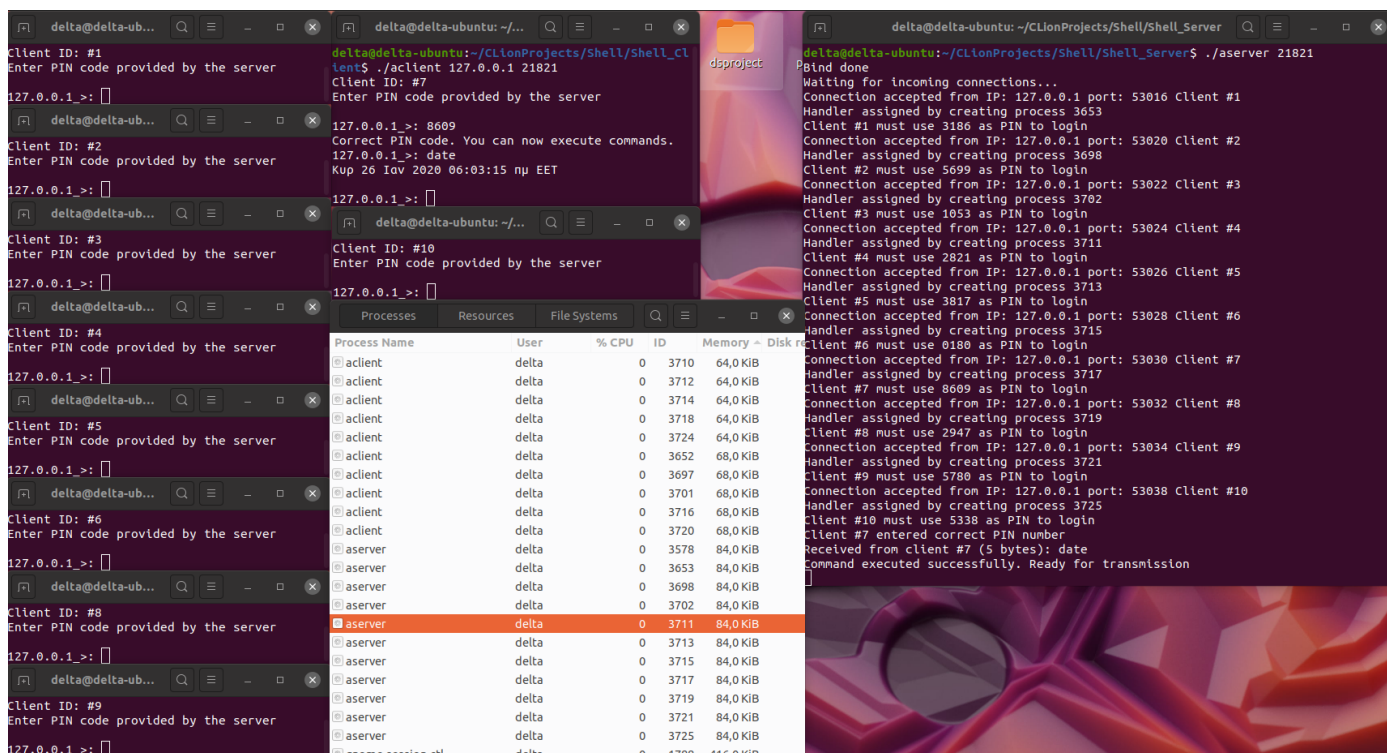
127.0.0.1_>: exit
Connection has been terminated.
delta@delta-ubuntu:~/CLionProjects/Shell/Shell_Client$
```

**Εικόνα 1:** Στιγμιότυπο βασικής επικοινωνίας μεταξύ του Server και των 2 Clients που εξυπηρετεί εκείνη τη στιγμή

### Παρατηρήσεις / Σχόλια

Ένα τυχαίο 4ψήφιο PIN δημιουργείται σε κάθε connection attempt μεταξύ Server/ Client. Ο κάθε Client παίρνει ένα δικό του μοναδικό ID, το οποίο θα βοηθήσει τον server admin να βρει τον σωστό κωδικό για να το πει στον χρήστη του Client.

## Υποστήριξη πολλών Client



**Εικόνα 2:** Στιγμιότυπο με 10 active clients. Ο Client #7 έβαλε σωστά το PIN του και μπορεί πλέον να εκτελεί εντολές. Παρατηρούμε ότι είναι ανοιχτές 11 διεργασίες Server: μία πατρική η οποία δέχεται τις νέες συνδέσεις ( PID: 3578 ) και άλλα 10 παιδιά-διεργασίες για να εξυπηρετούν τους 10 Clients.

### Παρατηρήσεις / Σχόλια

Υποστηρίζονται πολλαπλοί clients μέσω forking και κάλεσμα μιας συνάρτησης που έχει τον ρόλο του connection handler στο παιδί. Η γονική διεργασία πάντα θα υπάρχει για να δέχεται τις νέες συνδέσεις.

Ομαλή λειτουργία Server/Client (διαχείριση θυγατρικών διεργασιών - μη ύπαρξη Zombie - τερματισμός νέων διεργασιών στο κλείσιμο της επικοινωνίας με τον Client)

```
Client ID: #10
Enter PIN code provided by the server

127.0.0.1_>: 5338
Correct PIN code. You can now execute commands.
127.0.0.1_>: pwd
/home/delta/CLionProjects/Shell/Shell_Server
127.0.0.1_>: exit
Connection has been terminated.
delta@delta-ubuntu:~/CLionProjects/Shell/Shell_Server$

Handler assigned by creating process 3719
Client #8 must use 2947 as PIN to login
Connection accepted from IP: 127.0.0.1 port: 53034 Client #9
Handler assigned by creating process 3721
Client #9 must use 5780 as PIN to login
Connection accepted from IP: 127.0.0.1 port: 53038 Client #10
Handler assigned by creating process 3725
Client #10 must use 5338 as PIN to login
Client #7 entered correct PIN number
Received from client #7 (5 bytes): date
Command executed successfully. Ready for transmission
Client #8 disconnected : Address: 127.0.0.1 port: 53032
Client #8 exited abnormally.
Client #9 disconnected : Address: 127.0.0.1 port: 53034
Client #9 exited abnormally.
Client #6 disconnected : Address: 127.0.0.1 port: 53028
Client #6 exited abnormally.
Client #5 disconnected : Address: 127.0.0.1 port: 53026
Client #5 exited abnormally.
Client #7 disconnected : Address: 127.0.0.1 port: 53030
Client #7 exited abnormally.
Client #4 disconnected : Address: 127.0.0.1 port: 53024
Client #4 exited abnormally.
Client #3 disconnected : Address: 127.0.0.1 port: 53022
Client #3 exited abnormally.
Client #2 disconnected : Address: 127.0.0.1 port: 53020
Client #2 exited abnormally.
Client #1 disconnected : Address: 127.0.0.1 port: 53016
Client #1 exited abnormally.
Client #10 entered correct PIN number
Received from client #10 (4 bytes): pwd
Command executed successfully. Ready for transmission
Received from client #10 (5 bytes): exit
Client #10 requested exit. Process: 3725
Client #10 disconnected : Address: 127.0.0.1 port: 53038
Client #10 exited properly (exit).
```

| Process Name                    | User  | % CPU | ID   | Memory    | Disk I/O |
|---------------------------------|-------|-------|------|-----------|----------|
| aserver                         | delta | 0     | 3578 | 84,0 KiB  |          |
| at-spi2-registr...              | delta | 0     | 1778 | 632,0 KiB |          |
| at-spi-bus-launcher             | delta | 0     | 1762 | 1,0 MiB   |          |
| bash                            | delta | 0     | 3581 | 1,6 MiB   |          |
| bash                            | delta | 0     | 3023 | 1,6 MiB   |          |
| bash                            | delta | 0     | 3498 | 1,7 MiB   |          |
| dbus-daemon                     | delta | 0     | 1768 | 432,0 KiB |          |
| dbus-daemon                     | delta | 0     | 1609 | 1,5 MiB   |          |
| dconf-service                   | delta | 0     | 1821 | 896,0 KiB |          |
| evolution-addressbook-factor... | delta | 0     | 2091 | 3,7 MiB   |          |
| evolution-alarm-notify          | delta | 0     | 2047 | 15,0 MiB  |          |
| evolution-calendar-factory      | delta | 0     | 1961 | 4,5 MiB   |          |
| evolution-source-registry       | delta | 0     | 1882 | 4,3 MiB   |          |
| firefox                         | delta | 0     | 2366 | 158,7 MiB |          |
| gdm-x-session                   | delta | 0     | 1614 | 632,0 KiB |          |
| gimp-2.10                       | delta | 0     | 3259 | 98,4 MiB  |          |
| gnome-keyring-daemon            | delta | 0     | 1603 | 972,0 KiB |          |
| gnome-session-binary            | delta | 0     | 1640 | 1,8 MiB   |          |

**Εικόνα 3:** Και οι 10 από τους Clients του προηγούμενου παραδείγματος έκαναν exit είτε με Ctrl + C είτε μέσω της εντολής exit. Παρατηρούμε ότι στον System Monitor ( και άρα και στον Process Table ) παραμένει μονάχα η καταχώρηση της διεργασίας του Server που ελέγχει για νέες συνδέσεις ( PID : 3578 ). Ο handler της SIGCHLD κάνει καλή δουλειά.

### Παρατηρήσεις / Σχόλια

Σε κάθε κλήση της fork() που γίνεται μέσα στο πρόγραμμα, υπάρχει και μια waitpid() στον γονέα, η οποία αναμένει τον τερματισμό της διεργασίας του παιδιού και έπειτα λαμβάνει το exit code του παιδιού ώστε να μπορεί ο γονέας να πάρει αποφάσεις ανάλογα το αποτέλεσμα του παιδιού αυτού. Έτσι φροντίζουμε και race conditions αλλά και τα zombies. Η μοναδική εξαίρεση λαμβάνει χώρα στο forking για τους πολλαπλούς clients καθώς μία wait εκεί θα προκαλούσε τον server handler να μην δέχεται άλλα αιτήματα σύνδεσης μέχρι να τερματιστεί ο 1ος στη σειρά client. Η εξάλειψη των ζόμπι εδώ γίνονται εκμεταλλευόμενοι το σήμα SIGCHLD όπου μέσω της signal και ενός signal handler γίνεται το zombie harvesting.

## Υποστήριξη απλών εντολών και αναφορά σφαλμάτων

```
delta@delta-ubuntu:~/CLionProjects/Shell/Shell_Client$ ./aclient 192.168.1.2 21821
Client ID: #3
Enter PIN code provided by the server

192.168.1.2_>: 6368
Correct PIN code. You can now execute commands.
192.168.1.2_>: ls
aserver
compile.sh
Server.c
Server.h
tokenizer.c
tokenizer.h

192.168.1.2_>: pwd
/home/delta/Desktop/Project/Server

192.168.1.2_>: date
Sun Jan 26 22:46:50 EET 2020

192.168.1.2_>: df
Filesystem      1K-blocks    Used Available Use% Mounted on
udev             8092556         0   8092556   0% /dev
tmpfs            1623220      2276   1620944   1% /run
/dev/sda5        94813660 12500860 77453512 14% /
tmpfs            8116096    58484   8057612   1% /dev/shm
tmpfs             5120         4     5116   1% /run/lock
tmpfs            8116096         0   8116096   0% /sys/fs/cgroup
/dev/loop0       15104      15104         0 100% /snap/gnome-characters/296
/dev/loop1      153600    153600         0 100% /snap/gnome-3-28-1804/67
/dev/loop2       1024       1024         0 100% /snap/gnome-logs/61
/dev/loop3       4224       4224         0 100% /snap/gnome-calculator/406
/dev/loop4       3840       3840         0 100% /snap/gnome-system-monitor/100
/dev/loop5       91264     91264         0 100% /snap/core/8268
/dev/loop6      46080     46080         0 100% /snap/gtk-common-themes/1440
/dev/loop7       1024       1024         0 100% /snap/gnome-logs/81
/dev/loop8      160512    160512         0 100% /snap/gnome-3-28-1804/110
/dev/loop10      15104     15104         0 100% /snap/gnome-characters/375
/dev/loop11      90624     90624         0 100% /snap/core/7270
/dev/loop9       55808     55808         0 100% /snap/core18/1066
/dev/loop12      56064     56064         0 100% /snap/core18/1650
/dev/loop13     207872    207872         0 100% /snap/vlc/1397
/dev/loop14      4352       4352         0 100% /snap/gnome-calculator/544
/dev/loop15     45312     45312         0 100% /snap/gtk-common-themes/1353
/dev/loop16      3840       3840         0 100% /snap/gnome-system-monitor/123
/dev/loop17     141056    141056         0 100% /snap/code/23
tmpfs            1623216        16   1623200   1% /run/user/121
tmpfs            1623216        48   1623168   1% /run/user/1000
/dev/sde5       15067136 4581136 10486000 31% /media/delta/FILES
/dev/sde1       3626560    3626560         0 100% /media/delta/Kali Live2
/dev/sde2         716         696        20 98% /media/delta/Kali Live1
/dev/sde3      41017424 11153844 27766696 29% /media/delta/persistence
```

**Εικόνα 4:** Εκτέλεση απλών εντολών. Παρόλο που δεν το δείχνει η εικόνα αυτή, γίνεται και αναφορά σφαλμάτων.

### Παρατηρήσεις / Σχόλια

Εκτελούνται κανονικά απλές εντολές και γίνεται λεπτομερή αναφορά σφαλμάτων στον χρήστη.

## Υποστήριξη εντολών με παραμέτρους και ορίσματα και αναφορά σφαλμάτων

```
192.168.1.2_>: ls -alh
total 72K
drwxr-xr-x 2 delta delta 4.0K Jan 26 17:58 .
drwxr-xr-x 4 delta delta 4.0K Jan 26 17:54 ..
-rwxr-xr-x 1 delta delta 23K Jan 26 17:58 aserver
-rwxr-xr-x 1 delta delta 128 Jan 26 17:54 compile.sh
-rw-r--r-- 1 delta delta 21K Jan 26 17:54 Server.c
-rw-r--r-- 1 delta delta 599 Jan 26 17:54 Server.h
-rw-r--r-- 1 delta delta 3.8K Jan 26 17:54 tokenizer.c
-rw-r--r-- 1 delta delta 1.3K Jan 26 17:54 tokenizer.h

192.168.1.2_>: df -hT /home
Filesystem      Type  Size  Used Avail Use% Mounted on
/dev/sda5       ext4   91G   12G   74G   14% /

192.168.1.2_>: touch testfile

192.168.1.2_>: ls -l testfile
-rw-r--r-- 1 delta delta 0 Jan 26 22:49 testfile

192.168.1.2_>: rm testfile

192.168.1.2_>: ls -l testfile
execvp() executing ls : No such file or directory
```

**Εικόνα 5:** Εκτέλεση απλών εντολών με παραμέτρους. Χάρης στην ειδική δομή που έχει υλοποιηθεί, θεωρητικά μπορούν να υποστηριχθούν εντολές με τεράστιο αριθμό ορισμάτων.

### Παρατηρήσεις / Σχόλια

Εκτελούνται κανονικά απλές εντολές και γίνεται και λεπτομερή αναφορά σφαλμάτων στον χρήστη.



## Υποστήριξη πολλαπλών σωληνώσεων και αναφορά σφαλμάτων

```
delta@delta-ubuntu:~/CLionProjects/Shell/Shell_Client$ ./aclient 192.168.1.2 21821
Client ID: #7
Enter PIN code provided by the server

192.168.1.2_>: 3184
correct PIN code. You can now execute commands.
192.168.1.2_>: find /home/delta/Desktop -type f | grep Project | sort -r
/home/delta/Desktop/Project/Server/tokenizer.h
/home/delta/Desktop/Project/Server/tokenizer.c
/home/delta/Desktop/Project/Server/Server.h
/home/delta/Desktop/Project/Server/Server.c
/home/delta/Desktop/Project/Server/compile.sh
/home/delta/Desktop/Project/Server/aserver
/home/delta/Desktop/Project/Client/compile.sh
/home/delta/Desktop/Project/Client/Client.h
/home/delta/Desktop/Project/Client/Client.c
/home/delta/Desktop/Project/Client/aclient

192.168.1.2_>: find /home/delta/Desktop -type f | grep Project | wc -l
10
```

**Εικόνα 6:** Εκτέλεση εντολών που απαιτούν σωληνώσεις.

### Παρατηρήσεις / Σχόλια

Στην υλοποίηση αυτή μπορούν να εκτελεστούν εντολές που περιέχουν αόριστο αριθμό σωληνώσεων. Για κάθε σωλήνωση, απαιτείται ένα ζεύγος file descriptors καθώς και την κλήση συστήματος `pipe()`. Η κύρια ιδέα για την ακολουθιακή διαδικασία που απαιτείται για αυτήν την υλοποίηση πάρθηκε από αυτό το [Stack Overflow post](#). Μετά το τροποποίησα ώστε να είναι συμβατό με τη δομή που έφτιαξα ειδικά για τους σκοπούς της εργασίας (οι 3-star pointers θεωρούνται έτσι κι αλλιώς από μόνοι τους κακός C κώδικας).

Μια παραδοχή που αξίζει να σημειωθεί είναι ότι εδώ δεν γίνεται λεπτομερή αναφορά σφαλμάτων στον client στις περισσότερες περιπτώσεις αλλά επιστρέφεται μόνο το σφάλμα που αντιστοιχεί στο exit code της εντολής μέσω της `errno` (πχ. η `cat` μπορεί σε μη έγκυρο όνομα αρχείου να επιστρέψει `Operation Not Permitted` αντί για `No Such File Or Directory`). Αυτό έχει να κάνει με το γεγονός ότι εξαιτίας των πολλαπλών σωληνώσεων που γίνονται, θα έπρεπε να υπάρχει και ειδική διαχείριση στην `stderr` κάθε εντολής, πράγμα που θα περιέπλεκε αφάνταστα την διαδικασία. Πάντως σε ένα υποθετικό σενάριο, ο client καταλαβαίνει ότι κάτι δεν πήγε καλά μέσω του μηνύματος και θα μπορούσε να ρωτήσει τον διαχειριστή του server για την ακριβή αιτία του σφάλματος καθώς έχει την δυνατότητα να την κρατά στα logs του.

## Υποστήριξη απλών ανακατευθύνσεων και αναφορά σφαλμάτων

```
192.168.1.2_>: ls -alh | sort -hr
total 72K
-rwxr-xr-x 1 delta delta 23K Jan 26 17:58 aserver
-rwxr-xr-x 1 delta delta 128 Jan 26 17:54 compile.sh
-rw-r--r-- 1 delta delta 599 Jan 26 17:54 Server.h
-rw-r--r-- 1 delta delta 3.8K Jan 26 17:54 tokenizer.c
-rw-r--r-- 1 delta delta 21K Jan 26 17:54 Server.c
-rw-r--r-- 1 delta delta 1.3K Jan 26 17:54 tokenizer.h
drwxr-xr-x 4 delta delta 4.0K Jan 26 17:54 ..
drwxr-xr-x 2 delta delta 4.0K Jan 26 22:50 .

192.168.1.2_>: ls -alh | sort -hr > ls_res.txt
File made successfully.
192.168.1.2_>: ls -alh ls_res.txt
-rw-r--r-- 1 delta delta 420 Jan 27 00:05 ls_res.txt

192.168.1.2_>: cat ls_res.txt
total 72K
-rwxr-xr-x 1 delta delta 23K Jan 26 17:58 aserver
-rwxr-xr-x 1 delta delta 128 Jan 26 17:54 compile.sh
-rw-r--r-- 1 delta delta 599 Jan 26 17:54 Server.h
-rw-r--r-- 1 delta delta 3.8K Jan 26 17:54 tokenizer.c
-rw-r--r-- 1 delta delta 21K Jan 26 17:54 Server.c
-rw-r--r-- 1 delta delta 1.3K Jan 26 17:54 tokenizer.h
drwxr-xr-x 4 delta delta 4.0K Jan 26 17:54 ..
drwxr-xr-x 2 delta delta 4.0K Jan 26 22:50 .

192.168.1.2_>: ls -alh | sort -hr | wc -l > wc_res.txt
File made successfully.
192.168.1.2_>: cat wc_res.txt
10

192.168.1.2_>: Result shall be 11 now
execvp() executing Result : No such file or directory

192.168.1.2_>: ls -alh | wc -l
11

192.168.1.2_>: rm wc_res.txt

192.168.1.2_>: rm ls_res.txt

192.168.1.2_>: ls -alh
total 72K
drwxr-xr-x 2 delta delta 4.0K Jan 27 00:10 .
drwxr-xr-x 4 delta delta 4.0K Jan 26 17:54 ..
-rwxr-xr-x 1 delta delta 23K Jan 26 17:58 aserver
-rwxr-xr-x 1 delta delta 128 Jan 26 17:54 compile.sh
-rw-r--r-- 1 delta delta 21K Jan 26 17:54 Server.c
-rw-r--r-- 1 delta delta 599 Jan 26 17:54 Server.h
-rw-r--r-- 1 delta delta 3.8K Jan 26 17:54 tokenizer.c
-rw-r--r-- 1 delta delta 1.3K Jan 26 17:54 tokenizer.h
```

**Εικόνα 7:** Εκτέλεση εντολών με σωλήνωση και ανακατεύθυνση τους σε αρχείο εξόδου. Τα αρχεία αυτά αποθηκεύονται στον server.

### Παρατηρήσεις / Σχόλια

Υποστηρίζεται η απλή ανακατεύθυνση οποιουδήποτε συνδυασμού εντολών δηλαδή ακόμα και εντολών που περιέχουν πολλές σωληνώσεις. Ο μοναδικός περιορισμός είναι ότι η ανακατεύθυνση πρέπει να είναι πάντα στο τέλος της εντολής. Ανακατευθύνσεις ενδιάμεσα στις εντολές θα εξαναγκάσουν τον Server να στείλει το αποτέλεσμα της εντολής μόνο μέχρι την 1η μη έγκυρη ανακατεύθυνση. Με λίγο παραπάνω δουλειά είναι πιθανό να δουλέψουν και περιέργες συντάξεις εντολών με ενδιάμεσες ανακατευθύνσεις αλλά αφέθηκε με αυτήν την παραδοχή.



## Υποστήριξη cd, history και exit

```
delta@deltaUbuntu:~/Desktop/Project/Client$ ./aclient 127.0.0.1 21821
Client ID: #9
Enter PIN code provided by the server

127.0.0.1>: 7173
Correct PIN code. You can now execute commands.
127.0.0.1>: pwd
/home/delta/Desktop/Project/Server

127.0.0.1>: cd /home/delta
Successfully changed directory
127.0.0.1>: pwd
/home/delta

127.0.0.1>: cd
Wrong usage of cd/exit command
127.0.0.1>: cd /home/delta /home
Wrong usage of cd/exit command
127.0.0.1>: history
pwd
cd /home/delta
pwd
cd
cd /home/delta /home

127.0.0.1>: exit wonthappen
Wrong usage of cd/exit command
127.0.0.1>: exit
Connection has been terminated.
delta@deltaUbuntu:~/Desktop/Project/Client$
```

```
Connection accepted from IP: 127.0.0.1 port: 46242 Client #9
Handler assigned by creating process 6422
Client #9 must use 7173 as PIN to login
Client #9 entered correct PIN number
Received from client #9 (4 bytes): pwd
Command executed successfully. Ready for transmission
Received from client #9 (15 bytes): cd /home/delta
Received from client #9 (4 bytes): pwd
Command executed successfully. Ready for transmission
Received from client #9 (3 bytes): cd
Received from client #9 (21 bytes): cd /home/delta /home
Received from client #9 (16 bytes): exit wonthappen
Received from client #9 (5 bytes): exit
Client #9 requested exit. Process: 6422
Client #9 disconnected : Address: 127.0.0.1 port: 46242
Client #9 exited properly (exit).
```

**Εικόνα 8:** Χρήση των εντολών cd , history και exit. Η αριστερή εικόνα δείχνει τον client και η δεξιά τον sever.

### Παρατηρήσεις / Σχόλια

Υποστηρίζονται πλήρως οι cd και exit. Η history υλοποιήθηκε στην μεριά του client μετά από συζήτηση στο piazza καθώς προσωπικά μου φάνηκε πιο λογικό εκεί. Με λίγο πείραγμα στον κώδικα του server, ο sysadmin θα μπορούσε να κρατήσει logs με το τι εντολές εκτελούσε ποιος. Παράλληλα, δεν είναι ανάγκη για τον client να περιμένει την μετάδοση του ιστορικού του μέσω δικτύου.

## Γενικά Σχόλια / Παρατηρήσεις

---

Ήταν μία διασκεδαστική αλλά επίπονη εργασία, η οποία σε προκαλούσε όμως να την κάνεις να δουλεύει όσο καλύτερα μπορείς. Το παραπάνω αποτέλεσμα θα ήταν σχεδόν αδύνατο χωρίς τη χρήση εργαλείων όπως IDEs [ Visual Studio targeting WSL (Windows Subsystem for Linux) και CLion ] , Debuggers όπως τον GNU GDB για την απασφαλμάτωση του κώδικα και την κατανόηση συμπεριφοράς μερικών δύσκολων λειτουργιών και εννοιών και Memory Checkers όπως το Valgrind Memcheck όπου προειδοποιεί για memory leaks και παράνομες προσβάσεις στη μνήμη, οι οποίες πραγματικά μπορούν να καταστρέψουν μια C εφαρμογή. Τέλος, χρησιμοποιήθηκε και λίγο Wireshark , για να υπάρχει μια εκτίμηση του πως λειτουργούν τα sockets κοιτώντας τα από την προοπτική των TCP/IP στρωμάτων αλλά και για να ελέγξω αν τα δεδομένα ταξίδευαν στο δίκτυο ακριβώς με τον τρόπο που είχα ορίσει στην εφαρμογή.

Εξηγήσεις των πιο τεχνικών κομματιών του κώδικα μπορείτε να τις βρείτε στα source files. Αν θεωρείτε ότι πρέπει να δοθούν παραπάνω διευκρινήσεις, παρακαλώ επικοινωνήστε μαζί μου.

## Με δυσκόλεψε / Δεν υλοποίησα

---

Πολλά σημεία της υλοποίησης ήταν δύσκολα, όμως με υπομονή και σωστή επιλογή και χρήση εργαλείων εντέλει όλα υλοποιήθηκαν.

## Συνοπτικός Πίνακας

| 2η Εργασία   |                                  |                         |
|--|----------------------------------|-------------------------|
|  | Υλοποιήθηκε<br>(ΝΑΙ/ΟΧΙ/ΜΕΡΙΚΩΣ) | Συνοπτικές Παρατηρήσεις |
| client-server επικοινωνία με διαγνωστικά μηνύματα και έλεγχος τερματισμού επικοινωνίας<br>υποστήριξη πολλών client, PIN  | ΝΑΙ                              |                         |
| Ομαλή λειτουργία client server (διαχείριση θυγατρικών διεργασιών - μη ύπαρξη zombie - τερματισμός νέων διεργασιών στο κλείσιμο της επικοινωνίας με τον client) | ΝΑΙ                              |                         |
| Υποστήριξη απλών εντολών και αναφορά σφαλμάτων   | ΝΑΙ                              |                         |
| Υποστήριξη εντολών με παραμέτρους και ορίσματα και αναφορά σφαλμάτων   | ΝΑΙ                              |                         |
| Υποστήριξη απλών σωληνώσεων και αναφορά σφαλμάτων  | ΝΑΙ                              |                         |
| Υποστήριξη απλών ανακατευθύνσεων και αναφορά σφαλμάτων   | ΝΑΙ                              |                         |
| Υποστήριξη cd, history και exit  | ΝΑΙ                              |                         |