# CS341 Project 1 Report

Joonhyung Shin

September 11, 2017

## 1 Project Overview

This is the report for KAIST CS341 Project 1 by Joonhyung Shin. This report contains an instruction to build and run the program, the results of the self tests, and a brief explanation of how the program works.

## 2 Project Building

You can simply build the project by issuing `make` command. This automatically compiles and links the library code and source codes. Here, I provide a list of `make` targets. For more detail, see Section 4.

1. `make` or `make all`
   This target generates the static libraries, client executable, and server executables. As default, the `make` command is equivalent to the `make all` command. You will want to use `make` command most of the time.

2. `make client`
   This target generates the client executable file (and static libraries as well).

3. `make server`
   This target generates the server executable file that uses **fork()** function.

4. `make server_select`
   This target generates the server executable file that uses **select()** function.

5. `clean`
   This target removes all the executable files and the library object files.

6. Others
   There are also commands such as `make cs341proj.o` or `make libcs341proj.a` generates static library files and object files.

## 3 Project Running

The `make` commands generate three executable files: `client`, `server`, and `server_select`. The one of the differences with the project instruction is that there is `-d` option, which is a debug option. If this option is set, then the error messages will be printed to standard error. There are other differences, for instance, the host address is localhost, and the port number is 8000 by default.

# 4 Test Results

Note that the project was tested in Ubuntu 16.04 LTS. The project was tested via the following

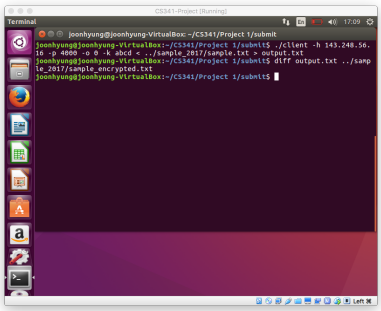

Figure 1: client test with remote server

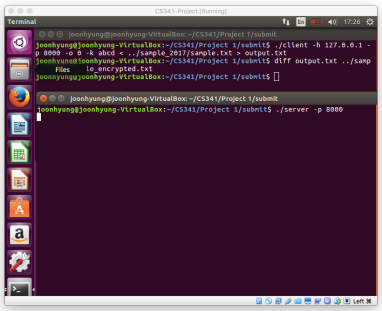

Figure 2: server test with local client

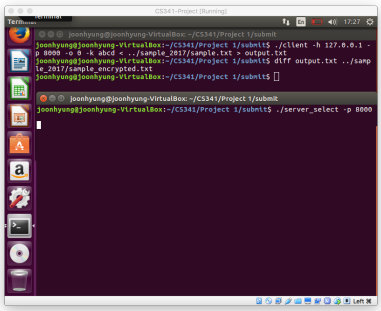

Figure 3: select server test with local client

commands. For details, please refer to Figure 1, 2, and 3.

1. `./client -h 143.248.56.16 -p 4000 -o 0 -k abcd`
2. `./server -p 8000`
3. `./server_select -p 8000`

# 5 Source Codes

This section briefly explains the important functions of source codes. There are 7 files including `Makefile`.

## 5.1 Source Tree

1. `cs341proj.h`
   This file includes the declarations of the functions in `cs341proj.c` and some frequently used macros.
2. `cs341proj.c`
   This file includes the implementations of frequently used functions by both client and server.
3. `client.c`
   This is the main client code.
4. `server_common.c`
   Since `server.c` and `server_select.c` share most of the functions, writing these functions in two separate files is redundant. This file contains common functions by two servers.
5. `server.c`
   This is the main server code implemented by `fork()`.
6. `server_select.c`
   This is the main server code implemented by `select()`.
7. `Makefile`
   This file helps building the project with `make` utility.

## 5.2 Functions

### 5.2.1 Common Functions

1. `int eprintf(const char *format, ...)`
   This function prints the given message in standard error prefixed with the executable file name. It is useful for debugging.

2. `uint16_t checksum(void *msg, size_t len)`
   This function calculates the checksum of the packet of length `len` with address `msg`.

3. `ssize_t read_packet(int fd, void *buf, size_t count)`
   This function robustly reads `count` bytes from the given socket file descriptor `fd`. This function reads until the length given by the header of the packet is been read.

4. `ssize_t write_packet(int fd, void *buf, size_t count)`
   This function robustly writes `count` bytes to the given socket file descriptor `fd`. This function only returns if an error occured or it successfully wrote `count` bytes.

### 5.2.2 Client

1. `int setup_client(void)`
   The client sets up the client-side file descriptor. It first calls `getaddrinfo()` function with arguments given by the user, and then tries to `connect()` to server after `socket()`. If it successes, then it returns the file descriptor.

2. `void start_client(int sockfd)`
   The client now communicates with server by sending packets using `read()` and `write()` functions. Here, we make use of `read_packet()` and `write_packet()` functions declared in `cs341proj.h`.

### 5.2.3 Server

1. `void reap_child(int signum)`
   This function reaps all the zombie processes. Since flagged `WNOHANG`, this function does not block.

2. `void handler_register(void)`
   The purpose of this function is to register `reap_child()` function as a signal handler of `SIGCHLD`. This helps us reap zombie processes, thus saves resources.

3. `int setup_server(void)`
   This function `bind()`s the process to a port provided by the user and ready to accept the connection requests from clients. It returns the listen file descriptor after the call of `listen()`.

4. `void cipher(int op, char *key_start, char *key_end, char *data, size_t len)`
   This function encrypts (or decrypts) data of length `len` with address `data`. Also, the key for Vigenere cypher is passed by the start address and the end address. Until this point, all the functions are defined in `server_common.c`.

5. `void start_server(int sockfd)`
   The function actually runs the server given the listen file descriptor. The server implemented by `fork()` forks the process each time the server accepts a client. The other

implemented by `select()` registers the listen file descriptor and connected file descriptors.

# 6 References


1. Randal E. Bryant, David R. O'Hallaron, 2016, Computer Systems A Programmer's Perspective, 3th edition, Pearson.

2. Beej's Guide to Network Programming, 2016, v3.0.21, Brian "Beej Jorgensen" Hall. `http://beej.us/guide/bgnet/output/html/singlepage/bgnet.html`.