

# Dipesh Kafle

## Question 1

1. Implement a port scanner using socket programming. The port scanner checks a number of ports (for instance, from 1 to 1026) to see if they are open (a server is listening on that port number) or closed (a server is not listening on that port number)

### Code

- port\_scanner.py

```
import socket,sys

def main():
    start = 1
    end = 1026

    addr = 'localhost'
    if len(sys.argv)>1 :
        addr = sys.argv[1]

    for port in range(start,end+1):
        try:
            sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
            sock.settimeout(1)
            res = sock.connect_ex((addr,port))
            if(res==0):
                print("{}: {} (tcp) is open".format(addr,port))
            sock.close()
        except:
            continue
main()
```

### Output

- I'm checking all the open ports of localhost. I ran the port\_scanner.py and it showed 80 and 631 are open. After running a command to see all

the ports open on my system, We can confirm that they are indeed 80 and 631.

```
LabNetworks/Lab2/Q1 on  cur_sem [!+?] via  v3.9.6
λ python port_scanner.py
localhost:80 (tcp) is open
localhost:631 (tcp) is open
LabNetworks/Lab2/Q1 on  cur_sem [!+?] via  v3.9.6
λ sudo lsof -i -P -n | grep LISTEN
[sudo] password for dipesh:
cupsd      412    root    7u  IPv6  17647      0t0  TCP [::1]:631 (LISTEN)
cupsd      412    root    8u  IPv4  17648      0t0  TCP 127.0.0.1:631 (LISTEN)
httpd      413    root    4u  IPv6  19591      0t0  TCP *:80 (LISTEN)
httpd      539    http    4u  IPv6  19591      0t0  TCP *:80 (LISTEN)
httpd      540    http    4u  IPv6  19591      0t0  TCP *:80 (LISTEN)
httpd      541    http    4u  IPv6  19591      0t0  TCP *:80 (LISTEN)
httpd      542    http    4u  IPv6  19591      0t0  TCP *:80 (LISTEN)
httpd      543    http    4u  IPv6  19591      0t0  TCP *:80 (LISTEN)
LabNetworks/Lab2/Q1 on  cur_sem [!+?] via  v3.9.6 took 2s
λ
```

## Question 2

2. Implement the following using TCP socket:

When the server receives a message from a client, it simply converts the message by using following rule

" If a character is a letter or a digit, it will be replaced with the next character in the character set, except that Z will be replaced by A, z by a and 9 by 0. Thus i becomes j, C becomes D, p becomes q and so on. Any character other than a letter or a digit will be replaced by a period(.)" and sends back the same to the client. This sending and receiving message should be done repeatedly until client and server send BYEBYE message.

## Code

- server.cpp

```
#include <algorithm>
#include <arpa/inet.h>
#include <iostream>
#include <netdb.h>
#include <netinet/in.h>
#include <numeric>
#include <optional>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
using namespace std;
```

```

void HANDLE_SEND_RECV_ERRORS(int st) {
    if (st == -1) {
        perror("Error in send()/recv()");
        exit(254);
    } else if ((st) == 0) {
        perror("Connection is closed because send/recv returned 0");
        exit(255);
    }
}

void die_with_error(const char *message, int err_code = 1) {
    perror(message);
    exit(err_code);
}

struct addrinfo *addr_setup(const char *address, const char *port_no) {
    struct addrinfo hints, *result;

    memset(&hints, 0, sizeof(hints));
    hints.ai_addr = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_flags = AI_PASSIVE;

    int status = getaddrinfo(address, port_no, &hints, &result);
    if (status != 0)
        die_with_error("getaddrinfo");
    return result;
}

int setup_socket(struct addrinfo *result) {
    int yes = 1;
    int sock_fd =
        socket(result->ai_family, result->ai_socktype, result->ai_protocol);
    if (sock_fd < 0)
        die_with_error("socket()");
    if (setsockopt(sock_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT | SO_LINGER,
        &yes, sizeof(yes)) < 0)
        die_with_error("setsockopt");
    if (bind(sock_fd, result->ai_addr, result->ai_addrlen) < 0)
        die_with_error("bind()");
    return sock_fd;
}

pair<int, struct sockaddr_storage> accept_connection(int sock_fd) {
    struct sockaddr_storage addr;
    socklen_t len = sizeof(addr);

```

```

    int client_fd = accept(sock_fd, (struct sockaddr *)&addr, &len);
    if (client_fd == -1)
        die_with_error("accept()");
    return {client_fd, addr};
}

void transform_buffer(string &s) {
    auto transformer = [](char c) {
        if (!isalnum(c))
            return '.';
        c += 1;
        if (c == 'z' + 1)
            c = 'a';
        else if (c == 'Z' + 1)
            c = 'A';
        else if (c == '9' + 1)
            c = '0';
        return c;
    };

    transform(s.begin(), s.end(), s.begin(), transformer);
}

void send_message(int conn, const string &message) {
    int sz = message.size();
    int htonl_sz = htonl(sz);

    int st = send(conn, &htonl_sz, sizeof(int), 0);
    HANDLE_SEND_RECV_ERRORS(st);

    st = send(conn, message.c_str(), sz, 0);
    HANDLE_SEND_RECV_ERRORS(st);
}

void handle_connection(int conn) {
    while (true) {
        int sz;
        int status = recv(conn, &sz, 4, 0);
        HANDLE_SEND_RECV_ERRORS(status);
        sz = ntohl(sz);

        char *buf = new char[sz + 1];
        status = recv(conn, buf, sz, 0);
        HANDLE_SEND_RECV_ERRORS(status);
        buf[status] = '\0';
    }
}

```

```

    string buffer(buf);
    cout << "Received : " << buffer << endl;
    delete[] buf;

    if (buffer == "BYEBYE") {
        send_message(conn, "BYEBYE");
        cout << "Sent : BYEBYE" << endl;
        break;
    } else {
        transform_buffer(buffer);
        send_message(conn, buffer);
        cout << "Sent : " << buffer << endl;
    }
}
}

int main() {
    struct addrinfo *result = addr_setup(NULL, "5000");
    int sock_fd = setup_socket(result);
    freeaddrinfo(result);

    listen(sock_fd, 10);

    auto [conn, addr] = accept_connection(sock_fd);
    handle_connection(conn);

    close(conn);
    close(sock_fd);
}

```

- client.cpp

```

#include <arpa/inet.h>
#include <iostream>
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

using namespace std;
void HANDLE_SEND_RECV_ERRORS(int st) {
    if (st == -1) {
        perror("Error in send()/recv()");
    }
}

```

```

        exit(254);
    } else if ((st) == 0) {
        perror("Connection is closed because send/rcv returned 0");
        exit(255);
    }
}

void die_with_error(const char *message, int err_code = 1) {
    perror(message);
    exit(err_code);
}

struct addrinfo *addr_setup(const char *address, const char *port_no) {
    struct addrinfo hints, *result;

    memset(&hints, 0, sizeof(hints));
    hints.ai_addr = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_flags = AI_PASSIVE;

    int status = getaddrinfo(address, port_no, &hints, &result);
    if (status != 0)
        die_with_error("getaddrinfo()");
    cout << "OK\n";
    return result;
}

int setup_client_socket(struct addrinfo *result) {
    int yes = 1;
    int sock_fd =
        socket(result->ai_family, result->ai_socktype, result->ai_protocol);
    if (sock_fd < 0)
        die_with_error("socket()");
    if (setsockopt(sock_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT | SO_LINGER,
        &yes, sizeof(yes)) < 0)

        die_with_error("setsockopt");

    if (connect(sock_fd, result->ai_addr, result->ai_addrlen) < 0)
        die_with_error("bind()");
    return sock_fd;
}

void send_message(int conn, const string &message) {
    int sz = message.size();
    int htonl_sz = htonl(sz);

```

```

    int st = send(conn, &htonl_sz, sizeof(int), 0);
    HANDLE_SEND_RECV_ERRORS(st);

    st = send(conn, message.c_str(), sz, 0);
    HANDLE_SEND_RECV_ERRORS(st);
}

string receive_message(int conn) {
    int sz;
    int status = recv(conn, &sz, 4, 0);
    HANDLE_SEND_RECV_ERRORS(status);
    sz = ntohl(sz);

    char *buf = new char[sz + 1];
    status = recv(conn, buf, sz, 0);
    HANDLE_SEND_RECV_ERRORS(status);
    buf[status] = '\0';

    string buffer(buf);
    cout << "Received : " << buffer << endl;
    delete[] buf;
    return buffer;
}

void handle_connection(int conn) {
    while (true) {
        string message;
        cout << "Enter message to send: ";
        cin >> message;
        cout.flush();

        send_message(conn, message);
        cout << "Sent : " << message << endl;

        string buffer = receive_message(conn);

        if (message == "BYEBYE" && buffer == "BYEBYE") {
            break;
        }
    }
}

int main() {
    struct addrinfo *result = addr_setup(NULL, "5000");
    int sock_fd = setup_client_socket(result);
}

```

```

    freeaddrinfo(result);
    handle_connection(sock_fd);
    close(sock_fd);
    return 0;
}

```

## Output

```

LabNetworks/Lab2/Q2 on p cur_sem [i+?]
λ ./server
Received : abcdefABCDEF
Sent : bcdefgBCDEFG
Received : hello
Sent : ifmp
Received : 1234567890
Sent : 2345678901
Received : azAZ
Sent : baBA
Received : abcd~!#$_-
Sent : bcde.....
Received : BYEBYE
Sent : BYEBYE
LabNetworks/Lab2/Q2 on p cur_sem [i+?] took 1m1s
λ

LabNetworks/Lab2/Q2 on p cur_sem [i+?]
λ ./client
OK
Enter message to send: abcdefABCDEF
Sent : abcdefABCDEF
Received : bcdefgBCDEFG
Enter message to send: hello
Sent : hello
Received : ifmp
Enter message to send: 1234567890
Sent : 1234567890
Received : 2345678901
Enter message to send: azAZ
Sent : azAZ
Received : baBA
Enter message to send: abcd~!#$_-
Sent : abcd~!#$_-
Received : bcde.....
Enter message to send: BYEBYE
Sent : BYEBYE
Received : BYEBYE
LabNetworks/Lab2/Q2 on p cur_sem [i+?] took 58s
λ

```

## Question 3

3. Assume that there two servers, A and B, which store a 10MB file that is split into 10 parts. Client 1 requests the file to server A which replies to client 1 with 5 chunks of the requested file, which are randomly selected. Later on, client 1 identifies the missing chunks and it requests the missing chunks (and only them) to server B. Moreover, the Client 1 can request 1 piece of chunk at the same. Once all the chunks are received, Client 1 sends the THANKS message to both of the servers.

## Code

- file\_splitter.h

```

#include <algorithm>
#include <fstream>
#include <iostream>
#include <iterator>
#include <map>
#include <numeric>
#include <queue>
#include <set>
#include <sstream>
#include <stack>
#include <string>
#include <string_view>

```



```

#include <unordered_map>
#include <unordered_set>
#include <vector>

using namespace std;

struct file_chunk {
    size_t sz;
    size_t chunk_no;
    string buf;
    file_chunk() {}
    file_chunk(size_t _sz, size_t _chunk_no, string _buf)
        : sz(_sz), chunk_no(_chunk_no), buf(_buf) {}
};

array<file_chunk, 10> splitter(const char *name) {
    fstream fp(name);
    stringstream strm;
    copy(istreambuf_iterator<char>(fp), istreambuf_iterator<char>(),
        ostreambuf_iterator<char>(strm));
    fp.close();
    string buffer = strm.str();

    int filesize = buffer.size();
    int one_chunk_size = filesize / 10;

    array<file_chunk, 10> chunks;

    auto it = buffer.begin();
    string tmp;
    for (int i = 0; i < 9; i++) {
        copy(it, it + one_chunk_size, back_inserter(tmp));
        it += one_chunk_size;
        chunks[i] = file_chunk(one_chunk_size, i, move(tmp));
    }
    copy(it, buffer.end(), back_inserter(tmp));
    chunks[9] = file_chunk(distance(it, buffer.end()), 9, move(tmp));
    return chunks;
}

```

- server\_A.cpp

```

#include "file_splitter.h"
#include <arpa/inet.h>
#include <iostream>
#include <netdb.h>
#include <netinet/in.h>

```

```

#include <optional>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

void HANDLE_SEND_RECV_ERRORS(int st) {
    if (st == -1) {
        perror("Error in send()/recv()");
        exit(254);
    } else if ((st) == 0) {
        perror("Connection is closed because send/recv returned 0");
        exit(255);
    }
}

void die_with_error(const char *message, int err_code = 1) {
    perror(message);
    exit(err_code);
}

struct addrinfo *addr_setup(const char *address, const char *port_no) {
    struct addrinfo hints, *result;

    memset(&hints, 0, sizeof(hints));
    hints.ai_addr = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_flags = AI_PASSIVE;

    int status = getaddrinfo(address, port_no, &hints, &result);
    if (status != 0)
        die_with_error("getaddrinfo");
    return result;
}

int setup_socket(struct addrinfo *result) {
    int yes = 1;
    int sock_fd =
        socket(result->ai_family, result->ai_socktype, result->ai_protocol);
    if (sock_fd < 0)
        die_with_error("socket()");
    if (setsockopt(sock_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT | SO_LINGER,
        &yes, sizeof(yes)) < 0)
        die_with_error("setsockopt");
}

```

```

    if (bind(sock_fd, result->ai_addr, result->ai_addrlen) < 0)
        die_with_error("bind()");
    return sock_fd;
}

pair<int, struct sockaddr_storage> accept_connection(int sock_fd) {
    struct sockaddr_storage addr;
    socklen_t len = sizeof(addr);
    int client_fd = accept(sock_fd, (struct sockaddr *)&addr, &len);
    if (client_fd == -1)
        die_with_error("accept()");
    return {client_fd, addr};
}

vector<int> get_five_random_nums(int mod) {
    srand(time(NULL));
    vector<int> nums(10);
    iota(nums.begin(), nums.end(), 0);
    random_shuffle(nums.begin(), nums.end());
    return vector<int>(nums.begin(), nums.begin() + 5);
}

void send_file_chunk(int conn, const file_chunk &chunk) {
    cout << endl << chunk.chunk_no << ":" << chunk.sz << endl;
    int size = chunk.sz;
    int chunk_id = chunk.chunk_no;
    int big_endian_sz = htonl(size);
    int big_endian_id = htonl(chunk_id);
    int status;

    status = send(conn, (void *)&big_endian_sz, sizeof(int), 0);
    cout << "sent size for " << chunk.chunk_no << ":" << status << endl;
    HANDLE_SEND_RECV_ERRORS(status);
    status = send(conn, (void *)&big_endian_id, sizeof(int), 0);
    cout << "sent id for " << chunk.chunk_no << ":" << status << endl;
    HANDLE_SEND_RECV_ERRORS(status);

    int total_sent = 0;
    const char *buf = chunk.buf.c_str();
    while (total_sent != size) {
        int bytes_sent =
            send(conn, (void *)(buf + total_sent), size - total_sent, 0);
        HANDLE_SEND_RECV_ERRORS(bytes_sent);
        cout << bytes_sent << ':' << total_sent << ' ';
        total_sent += bytes_sent;
    }
}

```

```

    cout << endl;
}

void handle_connection(int conn) {
    array<file_chunk, 10> file_chunks = splitter("./large_file.txt");
    auto five_random_chunk_no = get_five_random_nums(10);
    for (int id : five_random_chunk_no)
        cout << id << ' ' << file_chunks[id].buf.size() << '\n';
    for (int id : five_random_chunk_no) {
        // cout << id << '\n';
        send_file_chunk(conn, file_chunks[id]);
    }
}

int main() {
    int sock_fd;
    struct addrinfo *result = addr_setup(NULL, "3000");
    sock_fd = setup_socket(result);
    freeaddrinfo(result);
    listen(sock_fd, 10);

    auto [conn, addr] = accept_connection(sock_fd);
    handle_connection(conn);

    close(sock_fd);
}

...cpp
#include "file_splitter.h"
#include <arpa/inet.h>
#include <iostream>
#include <netdb.h>
#include <netinet/in.h>
#include <optional>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

void HANDLE_SEND_RECV_ERRORS(int st) {
    if (st == -1) {
        perror("Error in send()/recv()");
        exit(254);
    } else if ((st) == 0) {

```

```

        perror("Connection is closed because send/rcv returned 0");
        exit(255);
    }
}

void die_with_error(const char *message, int err_code = 1) {
    perror(message);
    exit(err_code);
}

struct addrinfo *addr_setup(const char *address, const char *port_no) {
    struct addrinfo hints, *result;

    memset(&hints, 0, sizeof(hints));
    hints.ai_addr = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_flags = AI_PASSIVE;

    int status = getaddrinfo(address, port_no, &hints, &result);
    if (status != 0)
        die_with_error("getaddrinfo");
    return result;
}

int setup_socket(struct addrinfo *result) {
    int yes = 1;
    int sock_fd =
        socket(result->ai_family, result->ai_socktype, result->ai_protocol);
    if (sock_fd < 0)
        die_with_error("socket()");
    if (setsockopt(sock_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT | SO_LINGER,
        &yes, sizeof(yes)) < 0)
        die_with_error("setsockopt");
    if (bind(sock_fd, result->ai_addr, result->ai_addrlen) < 0)
        die_with_error("bind()");
    return sock_fd;
}

pair<int, struct sockaddr_storage> accept_connection(int sock_fd) {
    struct sockaddr_storage addr;
    socklen_t len = sizeof(addr);
    int client_fd = accept(sock_fd, (struct sockaddr *)&addr, &len);
    if (client_fd == -1)
        die_with_error("accept()");
    return {client_fd, addr};
}

```

```

void send_file_chunk(int conn, const file_chunk &chunk) {
    cout << endl << chunk.chunk_no << ":" << chunk.sz << endl;
    int size = chunk.sz;
    int chunk_id = chunk.chunk_no;
    int big_endian_sz = htonl(size);
    int big_endian_id = htonl(chunk_id);
    int status;

    status = send(conn, (void *)&big_endian_sz, sizeof(int), 0);
    cout << "sent size for " << chunk.chunk_no << ":" << status << endl;
    HANDLE_SEND_RECV_ERRORS(status);
    status = send(conn, (void *)&big_endian_id, sizeof(int), 0);
    cout << "sent id for " << chunk.chunk_no << ":" << status << endl;
    HANDLE_SEND_RECV_ERRORS(status);

    int total_sent = 0;
    const char *buf = chunk.buf.c_str();
    while (total_sent != size) {
        int bytes_sent =
            send(conn, (void *)(buf + total_sent), size - total_sent, 0);
        HANDLE_SEND_RECV_ERRORS(bytes_sent);
        cout << bytes_sent << ':' << total_sent << ' ';
        total_sent += bytes_sent;
    }
    cout << endl;
}

void handle_connection(int conn) {
    array<file_chunk, 10> file_chunks = splitter("./large_file.txt");

    // ill keep taking requests always
    for (int i = 0; i < 5; i++) {
        int id;
        recv(conn, &id, sizeof(id), 0);
        id = ntohl(id);
        send_file_chunk(conn, file_chunks[id]);
    }
    char buf[100];
    int bytes_recvd = recv(conn, buf, 100, 0);
    buf[bytes_recvd] = 0;
    HANDLE_SEND_RECV_ERRORS(bytes_recvd);
    if (string(buf) == "THANKS") {
        cout << "Received : " << buf << '\n';
        return;
    } else {

```

```

        cerr << "Errorrrrrr" << endl;
        exit(255);
    }
}

int main() {
    int sock_fd;
    struct addrinfo *result = addr_setup(NULL, "4000");
    sock_fd = setup_socket(result);
    freeaddrinfo(result);
    listen(sock_fd, 10);

    auto [conn, addr] = accept_connection(sock_fd);
    handle_connection(conn);

    close(sock_fd);
}

```

- client.py

```

import socket
import sys
port = 3000
if(len( sys.argv )>=2):
    port = int(sys.argv[1])

def recv_data(sock , sz:int):
    recvd = 0
    BUF_SIZE=4096
    buffer = ''
    while((sz-recvd)>BUF_SIZE):
        x = sock.recv(BUF_SIZE)
        buffer+= x.decode()
        recvd= len(buffer)
    x = sock.recv(sz-recvd)
    buffer+=x.decode()
    return buffer

with socket.socket(socket.AF_INET,socket.SOCK_STREAM) as sock:
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR | socket.SO_REUSEPORT | socket.SO_REUSEPORT)
    sock.connect(('localhost',port))
    print(sock.getpeername())
    recvd_already = set()
    for i in range(5):
        sz = int.from_bytes(sock.recv(4),byteorder='big')

```

```

        id = int.from_bytes(sock.recv(4),byteorder='big')
        recvd_already.add(id)
        print(sz,id)
        open( "file{}.txt".format(id),"w" ).write(recv_data(sock,sz))
    sock.close()

not_recvd = set(range(0,10)) - recvd_already
sock2 = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
sock2.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR | socket.SO_REUSEPORT | socket.SO_REUSEPORT)
sock2.connect(('localhost',4000))

for id in not_recvd:
    sock2.send(id.to_bytes(4,byteorder='big'))
    sz = int.from_bytes(sock2.recv(4),byteorder='big')
    id = int.from_bytes(sock2.recv(4),byteorder='big')
    recvd_already.add(id)
    print(sz,id)
    open( "file{}.txt".format(id),"w" ).write(recv_data(sock2,sz))
sock2.send("THANKS".encode())
sock2.close()

```

## Output

```

LabNetworks/Lab2/Q3 on p cur_sem [!+] via v3.9.6 [17/1929]
A ./server_A
9:1000000
sent size for 9:4
sent id for 9:4
1000000:0
0:1000000
sent size for 0:4
sent id for 0:4
1000000:0
LabNetworks/Lab2/Q3 on p cur_sem [!+] via v3.9.6 [13/137]
A ./server_B
3:1000000
sent size for 3:4
sent id for 3:4
1000000:0
4:1000000
sent size for 4:4
sent id for 4:4
1000000:0
5:1000000
sent size for 5:4
sent id for 5:4
1000000:0

LabNetworks/Lab2/Q3 on p cur_sem [!+] via v3.9.6
A python client1.py
('127.0.0.1', 3000)
1000000 9
1000000 0
1000000 1
1000000 8
1000000 2
1000000 3
1000000 4
1000000 5
1000000 6
1000000 7
LabNetworks/Lab2/Q3 on p cur_sem [!+] via v3.9.6
A cat file0.txt file1.txt file2.txt file3.txt file4.txt f
file6.txt file7.txt file8.txt file9.txt > new_large_file.txt
LabNetworks/Lab2/Q3 on p cur_sem [!+] via v3.9.6
A diff large_file.txt new_large_file.txt
LabNetworks/Lab2/Q3 on p cur_sem [!+] via v3.9.6
<

```